Udacity Artificial Intelligence Nanodegree
Project – Implement a Planning Search
Heuristic Analysis

Introduction

The aim of this project is to solve the deterministic logistics planning problem for an air cargo transport system using a state-space planning search agent. The concept of tree searching has been adopted in the first and second projects of solving Sudoku and isolation games, and the same is adopted in this project. Indeed such problem is a classic example in the field of operations research. Mature research can be easily found in the literature of network optimization and dynamic optimization. The project is structured in Planning Domain Definition Language (PDDL), which facilitates the presentation and analysis of the search problem.

The problem is solved with planning graph and domain-independent heuristics with A* search. In this regard, planning graph [1] is a special data structure which is a polynomial-size approximation of the search tree of all possible actions, from the initial state to the successors, successors of those successors and so on. Planning graph is used because it reduces the amount of search needed to find the solution from the straightforward exploration of the state space graph. On the other hand, A* search algorithm is one of the graph- and tree- search algorithms which explores state space by generating successors of the already-explore states. As state and action are arranged alternatively in a planning graph, A* is particularly suitable for the search problem as such. Specifically, they consist of the follows:

- A* search with H1
  It is just uniform cost search as it always returns a value of one.
- A* search with “ignore preconditions” heuristics
  It estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed.
- A* search with “level-sum” heuristics
  It uses a planning graph representation of the problem state space to estimate the sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition.

The above would be compared with the following uniform non-heuristic search methods: 1) breadth first search, 2) breadth first tree search, 3) depth first graph search , 4) depth limited search, 5) uniform cost search, 6) recursive best first search H1 and 7) greedy best first graph search H1.

There are three kinds of objects – planes (p), airports (a) and cargoes (c), and three kinds of actions – load, unload and fly. The following outlines their 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))
```

Udacity Artificial Intelligence Nanodegree
Project – Implement a Planning Search
Heuristic Analysis

All the experiments were run on a workstation with Intel i7-5960X, 64GB ram and Ubuntu 16.04. In order to facilitate the output of the result, the script "run_search.py" is modified to the experimental result. For example, to output plan length and elapsed time, the following is added:

```
def show_solution(node, elapsed_time):
    print("Plan length: {}  Time elapsed in seconds: {}".format(len(node.solution()), elapsed_time))

    # Added to output plan length and elapsed time
    if not os.path.isfile(fileName):
        result = open(fileName, 'w')
    else:
        result = open(fileName, 'a')
    result.write(str(len(node.solution())) + ", ")
    result.write(str(elapsed_time) + "\n")
    result.close()

    for action in node.solution():
        print("{}{}".format(action.name, action.args))
```

Code to output problem name, algorithm expansions, goal tests, new nodes were accordingly added into "run_search.py".

Experimental Result for Air Cargo Problem 1

Initial state and goal 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))
```

Following is the experimental result:

Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Optimal?	Time (sec)
breadth_first_search	43	56	180	6	Yes	0.0692
breadth_first_tree_search	1458	1459	5960	6	Yes	0.9168
depth_first_graph_search	12	13	48	12	No	0.0071
depth_limited_search	101	271	414	50	No	0.0753
uniform_cost_search	55	57	224	6	Yes	0.0435
recursive_best_first_search with h_1	4229	4230	17029	6	Yes	2.3872
greedy_best_first_graph_search with h_1	7	9	28	6	Yes	0.0042
astar_search with h_1	55	57	224	6	Yes	0.0311
astar_search with h_ignore_preconditions	41	43	170	6	Yes	0.0313
astar_search with h_pg_levelsum	53	55	220	6	Yes	1.8110

Udacity Artificial Intelligence Nanodegree
Project – Implement a Planning Search
Heuristic Analysis

Experimental Result for Air Cargo Problem 2

Initial state and goal are:

```
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))
```

Following is the experimental result:

Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Optimal?	Time (sec)
breadth_first_search	3346	4612	30534	9	Yes	11.9084
breadth_first_tree_search	-	-	-	-	-	-
depth_first_graph_search	1124	1125	10017	1085	No	6.9680
depth_limited_search	213491	1967093	1967471	50	No	739.4151
uniform_cost_search	4853	4855	44041	9	Yes	10.6398
recursive_best_first_search with h_1	-	-	-	-	-	-
greedy_best_first_graph_search with h_1	998	1000	8982	21	No	2.1685
astar_search with h_1	4853	4855	44041	9	Yes	10.1459
astar_search with h_ignore_preconditions	1450	1452	13303	9	Yes	3.6444
astar_search with h_pg_levelsum	4390	4392	39972	9	Yes	1697.4854

Note: "-" means running time is more than 30 min and the experiment for that algorithm was interrupted.

Udacity Artificial Intelligence Nanodegree
Project – Implement a Planning Search
Heuristic Analysis

Experimental Result for Air Cargo Problem 3

Initial state and goal 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))
```

Following is the experimental result:

Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Optimal?	Time (sec)
breadth_first_search	14120	17673	124926	12	Yes	82.0075
breadth_first_tree_search	-	-	-	-	-	-
depth_first_graph_search	677	678	5608	660	No	3.1439
depth_limited_search	-	-	-	-	-	-
uniform_cost_search	18235	18237	159716	12	Yes	43.3880
recursive_best_first_search with h_1	-	-	-	-	-	-
greedy_best_first_graph_search with h_1	5614	5616	49429	22	No	13.5721
astar_search with h_1	18235	18237	159716	12	Yes	44.4450
astar_search with h_ignore_preconditions	5040	5042	44944	12	Yes	14.1361
astar_search with h_pg_levelsum	-	-	-	-	-	-

Note: "-" means running time is more than 30 min and the experiment for that algorithm was interrupted.

Udacity Artificial Intelligence Nanodegree
Project – Implement a Planning Search
Heuristic Analysis

Comparison and Analysis of Algorithms

The following table only lists algorithms that are optimal all three problems and can finish in 30 minutes.

Algorithm	Expansions			Goal Tests			New Nodes			Time (sec)		
	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
breadth_first_search	43	3346	14120	56	4612	17673	180	30534	124926	0.0692	11.9	82.0
uniform_cost_search	55	4853	18235	57	4855	18237	224	44041	159716	0.0435	10.6	43.4
astar_search with h_1	55	4853	18235	57	4855	18237	224	44041	159716	0.0311	10.1	44.4
astar_search with h_ignore_preconditions	41	1450	5040	43	1452	5042	170	13303	44944	0.0313	3.6	14.1

Out of the available 10 algorithms, only four of them can produce optimal solutions within 30 minutes, regardless of the number of expansions, goal tests, new nodes and computational time. A* search with "ignore preconditions" heuristics uses the least amount of computational time. This algorithm works by dropping all preconditions from actions and thus every action becomes applicable in every state. As the problem is more relaxed with the heuristics, the searching process can be speeded up as proven in this experiment. On the other hand, A* search with "ignore preconditions" heuristics has the least number of expansions and new nodes and is thus the most memory-efficient algorithm in this experiment.

In addition, A* search with H1 and uniform cost search are equivalent. Their value of expansions, goal tests and new nodes are the same, and their computational time is similar to each other. Indeed it is a surprise that uniform cost search works relatively competitive as compared to A* search with "ignore preconditions" heuristics in terms of CPU time – their CPU time is under the same order of magnitude. The same applies to breadth first search.

The following table lists the algorithms not listed in the above:

Algorithm	Expansions			Goal Tests			New Nodes			Plan Length			Time (sec)		
	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
breadth_first_tree_search	1458	-	-	1459	-	-	5960	-	-	6	-	-	6	-	-
depth_first_graph_search	12	1124	677	13	1125	678	48	10017	5608	12	1085	660	12	1085	660
depth_limited_search	101	213491	-	271	1967093	-	414	1967471	-	50	50	-	50	50	-
recursive_best_first_search with h_1	4229	-	-	4230	-	-	17029	-	-	6	-	-	6	-	-
greedy_best_first_graph_search with h_1	7	998	5614	9	1000	5616	28	8982	49429	6	21	22	6	21	22
astar_search with h_pg_levelsum	53	4390	-	55	4392	-	220	39972	-	6	9	-	6	9	-

Note: "-" means running time is more than 30 min and the experiment for that algorithm was interrupted.

Clearly, depth limited search is not memory efficient and takes up more than 1 million new nodes. Also, breadth first tree search, depth limited search, recursive best first search with H1 heuristics and A* search with "level sum" heuristics are not time-efficient, since they cannot finish at least one test case with 30 minutes.

Reference

[1] Russell S, Norving P. Artificial Intelligence. 3rd ed. New Jersey: Pearson; 2010.