HEURISTIC ANALYSIS

For The Planning Problem

SUBMITTED BY: AYUSH CHAUHAN

(Part of Artificial Intelligence Nanodegree)

Synopsis

In this project, I have developed a group of problems in classic PDDL (Planning Domain Definition Language) for the air cargo problem. The project was divided into two parts.

- Planning Problems

  In this part, I have developed methods in the AirCargoProblem class which act as a base for the air cargo planning problem. In this class I have implemented the following functions

  - get_actions() :- This method creates concrete actions (no variables) for all actions in the problem domain action schema and turns them into complete Action objects.
  - actions(state) :- Return the actions that can be executed in the given state.
  - result(state,action) :- Return the state that results from executing the given action in the given state. The action must be one of actions(state).

- Domain-independent heuristics

  In this part, I have developed methods for implementing the planning graph on the air cargo problem.

  - add_action_level (level) :- add an A (action) level to the Planning Graph
  - add_literal_level (level) :- add an S (literal) level to the Planning Graph
  - inconsistent_effects_mutex (a1,a2) :- Test a pair of actions for inconsistent effects, returning True if one action negates an effect of the other, and False otherwise.
  - interference_mutex(a1,a2) :- Test a pair of actions for mutual exclusion, returning True if the effect of one action is the negation of a precondition of the other.
  - competing_needs_mutex(a1,a2) :- Test a pair of actions for mutual exclusion, returning True if one of the precondition of one action is mutex with a precondition of the other action.
  - inconsistent_support_mutex(s1,s2) :- Test a pair of state literals for mutual exclusion, returning True if there are no actions that could achieve the two literals at the same time, and False otherwise.

Air Cargo 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))
```

Air Cargo Problem -1

Initital and goal state :

```
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))
```

Analysis of above problem using different uninformed and informed planning search techniques

Technique	Plan length	Time Taken (sec)	No of nodes expanded	Goal Test	New Nodes
breadth_first_search	6	1.36	43	56	180
breadth_first_tree_search	6	36.34	1458	1459	5960
depth_first_graph_search	12	0.31	12	13	48
depth_limited_search	50	2.71	101	271	414
uniform_cost_search	6	1.75	55	57	224
recursive_best_first_search with h_1	6	112.18	4229	4230	17029
greedy_best_first_graph_search with h_1	6	0.20	7	9	28
astar_search with h_1	6	0.29	55	57	224
astar_search with h_ignore_preconditions	6	0.23	41	43	170
astar_search with h_pg_levelsum	6	0.99	11	13	50

Optimal Path Length => 6

Optimal Path Length achieved by the technique marked in bold

Air Cargo Problem -2

Initital and goal state :

```
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))
```

Analysis of above problem using different uninformed and informed planning search techniques

Technique	Plan length	Time Taken (sec)	No of nodes expanded	Goal Test	New Nodes
breadth_first_search	9	100.27	3343	4609	30509
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	1669	58.50	1669	1670	14863
depth_limited_search	-	-	-	-	-
uniform_cost_search	9	132.60	4853	4855	44041
recursive_best_first_search with h_1	-	-	-	-	-
greedy_best_first_graph_search with h_1	17	27.07	998	1000	8982
astar_search with h_1	9	135.02	4853	4855	44041
astar_search with h_ignore_preconditions	9	42.21	1450	1452	13303
astar_search with h_pg_levelsum	9	86.38	86	88	841

Optimal Path Length => 9

Optimal Path Length achieved by the technique marked in bold

Air Cargo Problem -3

Initital and goal state :

```
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))
```

Analysis of above problem using different uninformed and informed planning search techniques

Technique	Plan length	Time Taken (sec)	No of nodes expanded	Goal Test	New Nodes
breadth_first_search	12	543.87	14663	18098	128605
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	195	111.68	3664	3665	29381
depth_limited_search	-	-	-	-	-
uniform_cost_search	12	558.38	18151	18153	157594
recursive_best_first_search with h_1	-	-	-	-	-
greedy_best_first_graph_search with h_1	26	193.48	6150	6152	53360
astar_search with h_1	12	554.96	18151	18153	157594
astar_search with h_ignore_preconditions	12	186.81	5038	5040	44751
astar_search with h_pg_levelsum	12	580.08	415	417	3782

Optimal Path Length => 12

Optimal Path Length achieved by the technique marked in bold

RESULT

For the above three problem set, breadth-first search and uniform cost search are only uninformed planning search technique that yields an optimal action plan. But both of them are computationally heavy as both require more memory and time for computation. On the basis of speed and memory, the depth-first graph is the best solution out of all uninformed searches.

All informed search strategies have yielded an optimal solution under 10 minute execution time limit but astar_search with h_pg_levelsum is the most optimal in case of memory management while astar_search with h_ignore_preconditions is the fastest among all the informed heuristic/search strategies.