Heuristic Analysis for Air Cargo Transport System

In this project, we have implemented a planning search agent to solve deterministic logistics planning problem for Air Cargo transport system. Searching is the universal technique of problem solving in AI. We use a planning graph and automatic domain-independent heuristics with A* search and compare their results with several non-heuristic search methods like breadth-first, depth-first, and uniform cost search.

Problems

We have three planning problems in the Air Cargo planning

Action (Load (c, p, a),

Precondition: At (c, a) ∧ At (p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

Effect: ¬ At (c, a) ∧ In (c, p))

Action (Unload (c, p, a),

Precondition: In (c, p) ∧ At (p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

Effect: At (c, a) ∧ ¬ In (c, p))

Action (Fly (p, from, to),

Precondition: At (p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

Effect: ¬ At (p, from) ∧ At (p, to))

Initial state and Goal state of the given 3 problems

Problem 1

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

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

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))

Breadth-First Search

It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

Disadvantage − since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

Disadvantage – Depth First Search algorithm may not terminate and go on infinitely on one path.

Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

Disadvantage − There can be multiple long paths with the cost ≤ C*. Uniform Cost search must explore them all.

*C** is the optimal path to the goal, the disadvantage of this algorithm is that it has to explore all the long paths with cost less than the optimal cost before finding the optimal path.

Heuristic Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

g(n) the cost (so far) to reach the node

h(n) estimated cost to get from the node to the goal

f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Problem 1				
Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	TRUE	6	0.03511	43
Depth First Search	FALSE	20	0.0169	21
Uniform Cost Search	TRUE	6	0.045	55
A * h_ignore_preconditions	TRUE	6	0.042	41
A * h_pg_levelsum	TRUE	6	1.45	11

Problem 2				
Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	TRUE	9	20.83	3343
Depth First Search	FALSE	619	5.985	624
Uniform Cost Search	TRUE	9	17.48	4853
A * h_ignore_preconditions	TRUE	9	5.96	1450
A * h_pg_levelsum	TRUE	9	272.03	86

Problem 3				
Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
Breadth First Search	TRUE	12	184.24	14663
Depth First Search	FALSE	392	3.541	408
Uniform Cost Search	TRUE	12	85.51	18223
A * h_ignore_preconditions	TRUE	12	25.208	5040
A * h_pg_levelsum			>15 min	

Search Strategy Discussion

The three non-heuristic search methods breadth first search, depth first search, and uniform cost search results are as follows.

Breadth first search considers the shortest path first and the result gives an acceptable amount of time and it is an optimal way.

Depth first results a less execution time and uses a less amount of memory, but it is lacks optimality, because it does not look for a better solution, it expands the nodes as deep as possible in the graph even though the goal is in its right branch.

Non-heuristic search performs better in problem 1 which says that when the problem size is small, using a method complex method like A* will increase in the solution complexity.

Heuristic based search performs well as the problem size is big. We can see this in problem 2 and problem 3, where A* h_ignore_preconditions search method outperformed the other methods.

From the above results and analysis we can say that, the breadth first method can solve planning problems both fast and optimality, which makes it a good candidate to start off an analysis when dealing with search planning problems, when the problem is small. When the problem have a huge search space, heuristic based methods like A* Search overshadows the performance of breadth first method.

Optimal paths for each problem:

Air Cargo Problem 1: **Breadth First Search:**

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Fly(P1, SFO, JFK)

Unload(C1, P1, JFK)

Air Cargo Problem 2: A* h_ignore_preconditions:

Load(C3, P3, ATL)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

Load(C2, P2, JFK)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Load(C1, P1, SFO)

Fly(P1, SFO, JFK)

Unload(C1, P1, JFK)

Air Cargo Problem 3: A* h_ignore_preconditions:

Load(C2, P2, JFK)

Fly(P2, JFK, ORD)

Load(C4, P2, ORD)

Fly(P2, ORD, SFO)

Unload(C4, P2, SFO)

Load(C1, P1, SFO)

Fly(P1, SFO, ATL)

Load(C3, P1, ATL)

Fly(P1, ATL, JFK)

Unload(C3, P1, JFK)

Unload(C2, P2, SFO)

Unload(C1, P1, JFK)