Planning Search Analysis

Chen Wen Shuo

This project is to solve deterministic logistics planning problems for an Air Cargo transport system using a planning search agent. With progression search algorithms like those in the navigation problem, optimal plans for each problem will be computed. Unlike the navigation problem, there is no simple distance heuristic to aid the agent. Instead, we implemented domain-independent heuristics.

Planning Domain Definitions Language problems

- **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))

- **Problem 1 initial State and Goal**

  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 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))

- **Problem 3 initial State and Goal**

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))

Uniformed Search Strategies Analysis

Uniformed Search have no addition information about state beyond that provided in the problem definition. In this section, we compare the performance of the uniformed search in terms of speed, memory usage and optimality.

Below is the code to run search algorithms to each problem.

Python run_search.py -p 1 -s 1 2 3 4 5 6 7

Python run_search.py -p 2 -s 1 3 5 7

Python run_search.py -p 3 -s 1 3 5 7

Problem 1 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth first	**6**	0.027	43	**Yes**
Breadth first tree	**6**	0.809	1458	**Yes**
Depth first graph	12	0.007	12	No
Depth limited	50	0.080	101	No
Uniform cost	**6**	0.031	55	**Yes**
Recursive best first	**6**	2.422	4229	**Yes**
Greedy best first	**6**	**0.004**	**7**	**Yes**

Problem 2 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth first	**9**	13.002	3343	**Yes**
Breadth first tree	-	-	-	
Depth first graph	575	3.093	**582**	No
Depth limited	-	-	-	
Uniform cost	**9**	11.769	4853	**Yes**
Recursive best first	-	-	-	
Greedy best first	21	**2.34**	998	No

Problem 3 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth first	**12**	94.901	14663	**Yes**
Breadth first tree	-	-	-	
Depth first graph	596	**2.825**	**627**	No
Depth limited	-	-	-	
Uniform cost	**12**	43.794	17783	**Yes**
Recursive best first	-	-	-	
Greedy best first	22	10.771	4031	No

Analysis

Among the uniformed search strategies, **Breadth First Search** and **Uniform Cost Search** are the only two uniformed search strategies that found the optimal plan length(6, 9, 12 for plan 1, plan 2, plan 3 respectively). When it comes to execution speed and memory usage, **Depth First Graph Search** is the fastest and uses the least memory. However, it does not generate an optimal action plan. If finding the optimal path length is important, we should use **Breadth First Search** or **Uniform Cost Search. Breadth First Search**

uses less memory while **Uniform Cost Search** runs faster. If finding the optimal path length is not the primary concern however, we should use **Greedy Best First Graph Search**. It is much faster than Breadth First Search and Uniform Cost Search. The downside is it does not always find the optimal path. In problem 1 and 2, it manages to find the optimal path. In problem 3, it does not find the optimal path but the path it generates is 22 instead of 12, which is much better than Depth First Search.

Informed Search Analysis

Informed search strategy can find solutions more efficiently than can an uninformed strategy. In this section, we compare the performance of **A* Search** using three different heuristics in terms of speed, memory usage and optimality.

Below is the code to run search algorithms to each problem.

Python run_search.py -p 1 -s 8 9 10

Python run_search.py -p 2 -s 8 9 10

Python run_search.py -p 3 -s 8 9 10

Problem 1 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
A* with h1 heuristic	**6**	**0.034**	55	**Yes**
A* with ignore preconditions heuristic	**6**	0.041	41	**Yes**
A* with level sum heuristic	**6**	0.703	**11**	**Yes**

Problem 2 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
A* with h1 heuristic	**9**	11.139	4853	**Yes**
A* with ignore preconditions heuristic	**9**	**5.238**	1450	**Yes**
A* with level sum heuristic	**9**	71.609	**86**	**Yes**

Problem 3 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
A* with h1 heuristic	**12**	49.790	17783	**Yes**
A* with ignore preconditions heuristic	**12**	**20.357**	5003	**Yes**
A* with level sum heuristic	**12**	351.488	**311**	**Yes**

Analysis:

All A* algorithms performed optimal plan. Among the informed search strategies, **A* with ignore preconditions heuristic** was the fastest, **A* with level sum heuristic** used the least node expansions, but the execution time is much slower.

Informed vs Uninformed Search Strategies

As we saw earlier, **Breadth First Search** is among the best uninformed search strategies, and **A* Search with Ignore Preconditions** heuristic is the best informed search strategies. So we are comparing these two search strategies.

Problem 1 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth First	**6**	**0.027**	43	**Yes**
A* with ignore preconditions heuristic	**6**	0.041	**41**	**Yes**

Problem 2 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth First	**9**	13.002	3343	**Yes**
A* with ignore preconditions heuristic	**9**	**5.238**	1450	**Yes**

Problem 3 Result

Search Strategy	Plan Length	Execution Time (sec)	Node Expansions	Optimal
Breadth First	**12**	94.901	14663	**Yes**
A* with ignore preconditions heuristic	**12**	**20.357**	5003	**Yes**

From the results above, **A* Search with Ignore Preconditions heuristic** is faster than **Breadth First Search** (significantly faster when the optimal length is over 9) and uses less memory. Therefore, **A* Search with Ignore Preconditions heuristic** would be the best choice overall for our Air Cargo problem.

Conclusion

Among all search strategies, we suggest to choose **A* Search with Ignore Preconditions heuristic** for Air Cargo Transport System in the efficiency, the speed, and the memory usage. The results clearly illustrate the benefits of using informed search strategies with custom heuristics over uninformed search techniques when searching for an optimal plan.