Optimal Sequence for the Three Cargo Problems

Problem 1

Problem initial state and goal is:

```
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))
```

Optimal Solution is:

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

Problem 2

Problem initial state and goal is:

```
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))
```

Optimal Solution is:

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
```

Problem 3

Problem initial state and goal is:

```
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))
```

Optimal Solution is:

```
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)

Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)
```

Search Comparisons

Given below is a table comparing various uninformed searches as well as A* with three heuristics.

Four uniformed searches were used across three problems. These searches were Breadth-First-Search(bfs), Depth First Search(dfs), Uniform Cost Search (ucs), Depth limited Serch (dls).

A* search was run across three problems with three heuristics which were a) h1 – it always returns one as heuristic value and hence is similar to Uniform Cost Search, b) levelsum – this heuristic returns the sum of level numbers at which each individual goal is met, and c) ignore_preconditions – which returns the number of actions required to be executed to attain all goals without worrying about the preconditions needed to execute the goal. It also ignores the situations where one action may negate a goal attained due to some other action earlier.

The table next page provides various statistics that were collected by running these search methods across three problems. The table is followed by an analysis of the results.

	Search Method	Expansions	Goal tests	New nodes	Time elapsed(sec)	Plan length	Optimality of solution
P1	bfs	43	56	180	0.0317	6	Yes
P1	dfs	21	22	84	0.0167	20	No
P1	ucs	55	57	224	0.0397	6	Yes
P1	depth limited search	101	271	414	0.0870	50	No
P1	A*_h1	55	57	224	0.0431	6	Yes
P1	A*_ignore_precond	41	43	170	0.0424	6	Yes
P1	A* levelsum	11	13	50	0.5396	6	Yes
P2	bfs	3346	4612	30534	13.6629	9	Yes
P2	dfs	107	108	959	0.3194	105	No
P2	ucs	4853	4855	44041	11.9557	9	Yes
P2	dls	Does not terminate					
P2	A*_h1	4853	4855	44041	11.6739	9	Yes
P2	A*_ignore_precond	1450	1452	13303	4.6215	9	Yes
P2	A* levelsum	86	88	841	45.7576	9	Yes
P3	bfs	14120	17673	124926	100.7464	12	yes
P3	dfs	292	293	2388	1.0934	288	No
P3	ucs	18223	18225	159618	52.5074	12	Yes
P3	dls	Does not terminate					
P3	A*_h1	18223	18225	159618	57.2000	12	Yes
P3	A*_ignore_precond	5040	5042	44944	20.8681	12	Yes
P3	A* levelsum	325	327	3002	259.9800	12	Yes

Uniformed Searches

As we can see from above, BFS always produces optimal solution as it searches one level of nodes before going deep in the graph to next level. However, the memory footprint of BFS is very high due the fact that fringe required to be maintained grows exponentially as we go deeper down the search graph. The number of nodes to be expanded also grow very fast. While BFS is a good (optimal and complete) search with problems having finite branching factors and uniform costs, it suffers from memory issues i.e. space complexity. These are evident from the table above, where BFS always found the optimal plan but with significantly higher number of node expansions and new nodes.

DFS is a fast search in the sense it needs to maintain only b nodes in fringe (b is the depth of tree). However, as it goes deeper into a branch, it may find a non optimal solution deep inside a branch while an optimal solution may exist on the next branch at a much shallow level closer to the root. For all the three problems above, DFS reach solution with lot lesser node expansions or fringe (new nodes), the solutions were not optimal in each case.

Third uninformed search was Uniform Cost Search (ucs). In the current setup it produces very similar results as BFS since the path cost is always one for each Action. UCS also produces optimal solution just life BFS. And it also suffers from same space complexity as BFS with number of node expansions and new node numbers being in the same range as that of BFS.

Fourth uninformed search was Depth Limited Search (dls) with depth limit(level) of 50. However, in our case the AIMA code for DLS returns 'cutoff' after hitting the maximum depth of 50. In our case DLS did not result in solution within the specified depth of 50 except for P1.

Iterative deepening (not covered here) which is a good approach to combine the benefits of both DFS and BFS especially in the problems where we need to return a solution in some fixed time. We start with shallow tree and use DFS to find a solution and then time permitting, we re-run the DFS search with tree being one level deeper than previous run.

A* Search

I ran A* search with three heuristics:

a) Heuristic always returning one – this is as good as UCS. This is also evident from the results above, where for all three problems the outcome of A* with h1 heuristic was similar to the results of UCS
b) Heuristic of ignore preconditions: This approach produced optimal solution for all three problems while reducing space complexity by a factor of 3 as compared to BFS. The time to run also came down by a factor of 3-5 depending on the size of problem with bigger problem P3 seeing maximum benefit as compared to BFS. While in our case it produced optimal solutions in all three problems, the use of greedy algorithm to set-cover issue does remove the guarantee of admissibility for this. This is explained in AIMA in 10.2.3. We are lucky to have optimal solution, this is not always guaranteed.
c) Heuristic of levelsum on a planning graph: This approach produced the most space efficient solution with minimum number of node expansions. However, the time to solution increased by a factor of 10 as compared to heuristic of "ignore preconditions". This heuristic is largely admissible and did produce optimal solution in our case. However, it can be inadmissible at times just like "ignore preconditions with greedy approach to set-cover issue".

A* with different heuristics shows the trade off of more accurate heuristics taking more time. So there is a balance that has to be drawn between accuracy and calculation-complexity of heuristics.

In overall analysis, I would settle for "A* with ignore precondition using greedy approach to set-cover" as the best strategy in this case. It reduced the space and time both as compared to BFS. Also a right balance between accuracy of heuristic and time as compared to A*_levelsum.