heuristic_analysis.md

Project 3 - Planning Search Problems

Metrics for planning solution searches

Before we start analyzing each problem with each of the search algorithms, let's define the `optimality` of a problem. First, we will find the optimal solution to the problem and describe the plan. The length of the optimal solution will be `len_opt_sol`. Then, the `optimality` is defined as: `len_opt_sol / len_current_search * 100`

Results for the **`air_cargo_p1`** problem.

Problem definition:

```
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:

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

Length of optimal solution: `len_opt_sol = 6`. | Algorithm | Expansions | Goal Tests | Time (s) | Optimality | | :-------------: | :--------------: | :----------: | :--------: | :--------: | | breadth_first_search | 43 | 56 | 0.053912139001113246 | 100% | | depth_first_graph_search | 21 | 22 | 0.022073595000620116 | 30% | | depth_limited_search | 101 | 271 | 0.15392913300092914 | 12% | | uniform_cost_search | 55 | 57 | 0.07226297700071882 | 100% | | astar_search (ignore_preconditions) | 41 | 43 | 0.05520928999976604 | 100% | | astar_search (level_sum) | 11 | 13 | 2.6240368370017677 | 100% |

Results for the **`air_cargo_p2`** problem.

Problem definition:

```
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:

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

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

Length of optimal solution: `len_opt_sol = 9`. | Algorithm | Expansions | Goal Tests | Time (s) | Optimality | | :-------------: | :-------------: | :---------: | :-------: | :-------: | | breadth_first_search | 3343 | 4609 | 24.304792158000055 | 100% | | depth_first_graph_search | 624 | 625 | 5.251924964000864 | 1.453% | | depth_limited_search | n/a | n/a | > 10 mins | n/a | | uniform_cost_search | 4852 | 4854 | 69.47413250000136 | 100% | | astar_search (ignore_preconditions) | 1506 | 1508 | 18.203693280000152 | 100% | | astar_search (level_sum) | 86 | 88 | 277.0997285960002 | 100% |

Results for the **`air_cargo_p3`** problem.

Problem definition:

```
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:

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)
```

Length of optimal solution: `len_opt_sol = 12`. | Algorithm | Expansions | Goal Tests | Time (s) | Optimality | | :-------------: | :-------------: | :---------: | :-------: | :-------: | | breadth_first_search | 14663 | 18098 | 161.9008968679991 | 100% | | depth_first_graph_search | 408 | 409 | 2.6251068919991667 | 3.061% | | depth_limited_search | n/a | n/a | > 10 mins | n/a | | uniform_cost_search | n/a | n/a | > 10 mins | n/a | | astar_search (ignore_preconditions) | 5118 | 5120 | 113.36011673600296 | 100% | | astar_search (level_sum) | n/a | n/a | > 10 mins | n/a |

Comments

Something that we can see right from the start is that `astar_search` and `breadth_first_search` always produce 100% optimal solutions. This is due to the fact that BFS, by design, searches for the shortest path within a planning graph, and `astar_search` is built on top of it as well. Moreover, it is interesting to observe the timing differences between these most optimal search results - `astar_search (level_sum)` produces the most optimal result in *the longest time*, almost 10 times slower or more in some instances of the search space. This is a result of the overhead implied by calculating a heuristic at each step. We also notice that `astar_search (ignore_preconditions)` searches a significantly reduce number of nodes compared to simple BFS, in roughly the same amount of time! So, reducing the complexity of the problem, by ignoring the actions' preconditions, does help in this situation.

Overall, in the case of the Air Cargo problem, it seems like `astar_search (ignore_preconditions)` offers the most reliable results in terms of optimality, time of execution and number of expanded nodes. This might not always be the case though - as we can see, as the problem becomes small enough, a simple BFS might be a more straight-forward approach.