






 safdark / AI-planning Private




 Unwatch 1  Star 0  Fork 0

 Code  Issues 0  Pull requests 0  Projects 0  Wiki  Settings Insights

Branch: master AI-planning / README.md Find file Copy path

 safdark Update README.md 0ec962d 6 minutes ago

4 contributors    

158 lines (106 sloc) 12.3 KB  Raw  Blame  History   

	Heuristic			Problem # 1					Problem # 2					Problem # 3				
	H-1	Ignore Preconditions	Level Sum - Planning Graph	Expansions	Goal Tests	New Nodes	Plan Length	Time	Expansions	Goal Tests	New Nodes	Plan Length	Time	Expansions	Goal Tests	New Nodes	Plan Length	Time
1 Breadth First Search	-	-	-	43	56	180	6	0.033	3343	4609	30509	9	12.569	14663	18098	129631	12	92.651
2 Breadth First Tree Search	-	-	-	1458	1459	5960	6	0.892	> 10 mins					> 10 mins				
3 Depth First Graph Search	-	-	-	21	22	84	20	0.013	624	625	5602	619	3.178	408	409	3364	392	1.592
4 Depth Limited Search	-	-	-	101	271	414	50	0.087	> 10 mins					> 10 mins				
5 Uniform Cost Search	-	-	-	55	57	224	6	0.035	4853	4855	44041	9	11.12	18162	18164	159128	12	49.596
6 Recursive Best First Search	X	-	-	4229	4230	17023	6	2.978	> 10 mins					> 10 mins				
7 Greedy Best First Graph Search	X	-	-	7	9	28	6	0.004	998	1000	8982	17	2.52	5380	5382	47545	26	15.761
8 A* Search	X	-	-	55	57	224	6	0.044	4853	4855	44041	9	12.19	18162	18164	159128	12	53.821
9 A* Search	-	X	-	41	43	170	6	0.044	1450	1452	13303	9	4.452	5038	5040	44924	12	17.321
10 A* Search	-	-	X	50	52	208	6	1.147	> 10 mins					> 10 mins				

([https://github.com/safdark/AI-planning/blob/master/Project3\\_Data.png](https://github.com/safdark/AI-planning/blob/master/Project3_Data.png))

# Path Planning - Air Cargo Transport

This document gives a high level overview of the search algorithms being evaluated, and a description of the problems they are being evaluated against.

## Overview

### Common Terms

*Runtime Metrics:* References to the word 'Performance Metrics' will be intended to refer to an algorithm's performance across 4 metrics - runtime, node expansions, goal tests and new node creations. This avoids redundancy

*Frontier:* This is a collection of the nodes whose neighbors have not yet been explored. In other words, these nodes have been reached, but have not been explored further. For the purpose of our discussion, the 'frontier' also encapsulates the underlying *representation* of this collection -- i.e, either a list, stack, queue, priority-queue etc. This is important to consider here because the choice of this data structure precisely determines the type of search algorithm ultimately being implemented (eg. Breadth-First-Search, Depth-First-Search, Uniform-Cost-Search etc), because this data structure dictates the order in which elements are selected at each subsequent step of the search algorithm.

### Problem Structures

*Tree-Search:* The structure of the search space in this case is a tree, with no cross-edges or back-edges (as is possible with a graph search). The implication of this on the algorithm is that there is no need to maintain a list of explored/visited nodes. Each neighbor of a node that is being explored/visited is guaranteed not to have been visited before.

*Graph-Search:* The structure of the search space in this case is a graph, with forward-edges, cross-edges and back-edges. The implication of this on the search algorithm is that it needs to maintain a list of explored/visited edges, and each time an edge's neighbors are explored, those neighbors that have already been explored are not added to the frontier. The Graph-Search algorithm reduces to a Tree-Search algorithm if the graph being searched on fulfills the structural requirements of being a tree data structure (i.e, no back- or cross- edges).

## Search Algorithms Used

### Non-heuristic

*Breadth-First-Search:* A graph-based search algorithm wherein the next node visited from the frontier is chosen based on the MIN # of hops from the start. Frontier = regular queue (FIFO).

*Breadth-First-Tree-Search:* A graph-based search algorithm wherein the next node visited from the frontier is chosen based on the MIN # of hops from the start. Frontier = regular queue (FIFO).

*Depth-First-Graph-Search:* A graph-based search algorithm wherein the next node visited from the frontier is chosen based on the MAX # of hops from the start. Frontier = regular stack (FILO).

*Depth-Limited-Search:* A variant of a depth-first-graph-search wherein the search is "cut-off" (i.e, back-tracks and continues to the next neighbor of the parent node) if the given node is at a depth that is lower than the cut-off depth that is provided as a parameter to the search. This can prevent an algorithm from ratholing too far down a search path by pruning out lengthy search paths.

*Uniform Cost Search:* A graph-based search algorithm wherein the next node visited from the frontier is chosen based on the MIN *total-path-cost* of the node. Total-path-cost is the cost of reaching that node from the start node. A Uniform Cost Search reduces to a Breadth-First-Search when all graph edges have the same cost (eg. cost of 1). Frontier = priority queue.

### Heuristic

*Terms:*

- $f(n)$  = Function that is to be minimized when selecting the next 'n'.
- $g(n)$  = Cost of reaching the node n from the start
- $h(n)$  = Estimated cost of reaching the goal from the node n.

*Heuristics:*

- $H_1$ : Considers  $f(n) = 1$ .
- $H_{\text{Ignore\_Preconditions}}$ : This heuristic estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed. In other words,  $f(n) = \#$  of unsatisfied goals remaining at the node n.
- $H_{\text{LevelSum}}$ :

*Best-First Search:* Search the nodes with the lowest f scores first. You specify the function  $f(\text{node})$  that you want to minimize; for example, if f is a heuristic estimate TO the goal, then we have greedy best first search; if f is node.depth then we have breadth-first search.

*Greedy-Best-First-Graph-Search:* A graph-based search algorithm wherein the next node visited from the frontier is chosen based on the MIN of the *estimated* cost of the node to the goal.

*A-Star Search:* A-Star search is best-first graph search with  $f(n) = g(n)+h(n)$ .

## Environment Characteristics

The planning problem being considered has the following characteristics:

- Observability: Full (vs Partial)
- Agent: Single (vs Multi-Agent)

- Determinism: Deterministic (vs Stochastic)
- Progression: Sequential (not Episodic)
- Dynamism: Static (not Dynamic)
- Time: Discrete (not Continuous)
- Knowledge: Known (not Unknown)

## Problem Definition

A summary of the "Air Cargo Transport" problem, in Planning Domain Definition Language (PDDL).

- 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, SF0) ∧ At(C2, JFK)
  ∧ At(P1, SF0) ∧ At(P2, JFK)
  ∧ Cargo(C1) ∧ Cargo(C2)
  ∧ Plane(P1) ∧ Plane(P2)
  ∧ Airport(JFK) ∧ Airport(SF0))
Goal(At(C1, JFK) ∧ At(C2, SF0))
```

- Problem 2 initial state and goal:

```
Init(At(C1, SF0) ∧ At(C2, JFK) ∧ At(C3, ATL)
  ∧ At(P1, SF0) ∧ At(P2, JFK) ∧ At(P3, ATL)
  ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
  ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
  ∧ Airport(JFK) ∧ Airport(SF0) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SF0) ∧ At(C3, SF0))
```

- Problem 3 initial state and goal:

```
Init(At(C1, SF0) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
  ∧ At(P1, SF0) ∧ At(P2, JFK)
  ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
  ∧ Plane(P1) ∧ Plane(P2)
  ∧ Airport(JFK) ∧ Airport(SF0) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SF0) ∧ At(C4, SF0))
```

## Optimal Plan

Overall, 'A-Star with the ignore-preconditions heuristic' performed with the best runtime metrics - fastest runtime, fewer node expansions, fewer goal tests, and fewer new node creations - among the search algorithms used, while also yielding an optimal path in each case.

The contrast in performance as compared to non-heuristic searches is reflected especially so in larger problems, such as problem # 2 and problem # 3. This is understandable, since non-heuristic searches grow exponentially as we

expand the frontier of the search at each step. Smaller problems, however, (such as problem # 1) are more amenable to the non-heuristic search methods, making the difference between the two less obvious.

## Comparison of non-heuristic search result metrics (optimality, runtime, node expansions)

---

Among the non-heuristic-based search algorithms, a simple Breadth First Search seems to have achieved optimal performance both in terms of the plan length and runtime metrics.

### Least Optimal: Depth First Search

DFS appears to have created the least optimal plans (by a significant margin), amongst all the other search algorithms, across all the 3 problems.

The reason for sub-optimal results is that the algorithm picks the first available plan connecting the initial state to the goal state, regardless of its optimality. It focuses on finding "a" planning path, rather than finding one of "the" paths from initial to goal states. For the same reason, the expansions, goal tests, new nodes and runtime for this search algorithm applied to problem # 3 were lower than that for problem # 2, even though the size difference might have suggested the opposite. In this particular case, it is likely that the transitions from initial to goal states for problem # 3 might have been clustered closer to the side of the graph that the depth first search expanded first.

However, that may not always be the case. It is possible to conceive of a scenario wherein all valid paths might be clustered near the side of the graph that gets expanded last by this algorithm. In that scenario, Best-First-Search-based searches would likely yield better runtime metrics than the DFS approaches, thereby beating DFS both in plan optimality as well as runtime metrics.

### Most Optimal

#### *Breadth First Search:*

The issue seen with DFS's non-determinism and resulting sub-optimality is not found with breadth-first search since all nodes at a given level would be visited before going to the next level, thereby assuring that if a path existed to a goal, the most straightforward/optimal/shortest such path would be discovered, rather than just "a" possible path of any length. In the case of the air cargo problem, wherein each edge is of the same cost, BFS is assured to yield an optimal solution.

BFS's runtime metrics were worse than that of DFS, but that is not necessarily the norm, and therefore cannot be considered in this comparative analysis beyond what has already been discussed in the section above.

#### *Uniform Cost Search:*

Though UCS achieved an optimal plan for all 3 problems, the runtime metrics were mildly worse than those achieved by BFS in each case.

## Comparison & contrast of A-Star heuristic search result metrics

---

In all three problems, the heuristic-based A-Star searches that eventually completed, all yielded an optimal path of the same length as the optimal path generated by the non-heuristic search results. Below are the relative results using each heuristic:

- Ignore preconditions: In all 3 problems, A-Star with the ignore-preconditions heuristic yielded optimal results in a highly performant manner.
- Level sum: A-Star with the level-sum heuristic took > 10 mins for problems 2 and 3. Though it yielded an optimal solution for problem 1, it did not beat the performance of A-Star with the ignore-preconditions heuristic for the same problem.

In the specific case of problem # 1, the greedy-best-first-graph search algorithm achieved an optimal plan with the best runtime metrics. However, for problems 2 and 3, it performed far worse than A-Star. Therefore, in general, the A-

Star search with the ignore-precondition heuristic not only achieved optimal plans, but also did that more efficiently than the non-heuristic approaches, as can be expected of heuristic-based searches.

## What was the best heuristic?

A-Star with the ignore-preconditions heuristic, was the best performing path planning search agent across the domain of non-heuristic-based and heuristic-based algorithms, barring an exceptional performance by the greedy-best-first-search algorithm for problem 1. Considering however, that problem 1 was smaller and more amenable to an exhaustive search, A-Star with ignore-preconditions seems to shine far more in cases where the problem has a larger state space and action schema.

## Heuristic vs Non-heuristic search comparison

In general, the non-heuristic-based search algorithms guarantee optimal planning, but at heavy performance cost. For instance, for both problems 2 and 3, three of the non-heuristic based algorithms (i.e, Breadth First Tree Search, Depth Limited Search, and Recursive Best First Search) took > 10 mins. The only non-heuristic-based search algorithms to not guarantee optimal planning were the Greedy-Best-First-Graph-Search, the Depth-First-Graph-Search and the Depth-Limited-Search algorithms.

Heuristic-based search algorithms, on the other hand, tended to yield an optimal plan for all three problems, with the exception of the A-Star with level-sum heuristic, that took > 10 mins for problems 2 and 3. Considering that 3 out of the 6 non-heuristic-based search algorithms took > 10 mins for problems 2 and 3, and only 1 out of the 4 heuristic-based search algorithms did the same, heuristic-based searches achieved pretty good performance improvement without sacrificing optimal planning.

