Heuristic Analysis

Dovydas Čeilutka

January 8, 2018

In this paper the three planning problems are tackled using the uninformed search strategies and then using heuristic (informed) search strategies. The search strategies are evaluated by their ability to find the optimal plan, speed and memory-efficiency. The performance is then compared between and within the groups of search strategies. Moreover, the strategies are used to find optimal plans for the problems.

Software Due to the nature of the exercise the quality of the hardware and the efficiency of the software have major impact on the measurements. A rather minor part of the software was coded by the autor of this paper with the other portion of the software being provided by Udacity. Python 3.6 on macOS High Sierra is used for running all the scripts.

Hardware As for the hardware the code was run on a modern laptop with 2.7 GHz Intel Core i7 processor and 16 GB of RAM.

1 Optimal plans

The optimal plans for the three problems were found by running `run_search.py` script using breadth-first search. The breadth-first search was chosen, because it is complete and optimal, thus it is guaranteed to eventually find the optimal solution (Russell and Norvig 2010).

1.1 Optimal plan for Problem 1

The optimal plan for Problem 1 is:

1. Load(C1, P1, SFO)
2. Load(C2, P2, JFK)
3. Fly(P2, JFK, SFO)
4. Unload(C2, P2, SFO)
5. Fly(P1, SFO, JFK)
6. Unload(C1, P1, JFK)

The plan has a length of 6.

1.2 Optimal plan for Problem 2

The optimal plan for Problem 2 is:

1. Load(C2, P2, JFK)
2. Load(C1, P1, SFO)
3. Load(C3, P3, ATL)

4. Fly(P2, JFK, SFO)
5. Unload(C2, P2, SFO)
6. Fly(P1, SFO, JFK)
7. Unload(C1, P1, JFK)
8. Fly(P3, ATL, SFO)
9. Unload(C3, P3, SFO)

The plan has a length of 9.

1.3 Optimal plan for Problem 3

The optimal plan for Problem 3 is:

1. Load(C2, P2, JFK)
2. Load(C1, P1, SFO)
3. Fly(P2, JFK, ORD)
4. Load(C4, P2, ORD)
5. Fly(P1, SFO, ATL)
6. Load(C3, P1, ATL)
7. Fly(P1, ATL, JFK)
8. Unload(C1, P1, JFK)
9. Unload(C3, P1, JFK)
10. Fly(P2, ORD, SFO)
11. Unload(C2, P2, SFO)
12. Unload(C4, P2, SFO)

The plan has a length of 12.

2 Uninformed search strategies

In this section these four uninformed (blind) search strategies are compared in their ability to solve problems 1 - 3:

1. Breadth-first search
2. Uniform-cost search
3. Depth-first graph search
4. Depth-limited search

2.1 Comparison of the uninformed search strategies

Russell and Norvig 2010 suggest evaluating the performance of the search strategies using these four criteria:

1. Completeness: Is the algorithm guaranteed to find a solution when there is one?
2. Optimality: Does the strategy find the optimal solution?
3. Time complexity: How long does it take to find a solution?
4. Space complexity: How much memory is needed to perform the search?

Breadth-first search is an instance of the general graph-search algorithm which expands the shallowest nodes first. It is **complete** if b is finite and **optimal** if step costs are identical. It has exponential time complexity $O(b^d)$ and exponential space complexity $O(b^d)$, where b is a branching factor and d is the depth (Russell and Norvig 2010).

Uniform-cost search is an extension of the breadth-first search strategy, which is **optimal** for any step-cost function. The uniform-cost strategy expands not the shallowest node, but the node with the lowest path cost and does the goal test when the node is selected for expansion. The **completeness** characteristic is retained. It has exponential time complexity $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ and exponential space complexity $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ where b is a branching factor and $C*$ is a cost of the optimal solution and ϵ is the lowest allowed cost of a path (required to avoid getting stuck in an infinite series of 0-cost steps) (Russell and Norvig 2010).

Depth-first graph search expands the deepest node first. It is **complete** in finite state spaces, but not optimal. Time and space complexity remain the same as in the breadth-first search: time complexity is $O(b^d)$ and the space complexity is $O(b^d)$, where b is a branching factor and d is the depth (Russell and Norvig 2010).

Depth-limited search is an improvement on the depth-first (graph) search that has a predetermined depth limit l. The nodes below the l level are ignored. It is not complete and not optimal. Time complexity is $O(b^l)$ and the space complexity is $O(b^l)$, where b is a branching factor and l is the cutoff depth (Russell and Norvig 2010).

2.2 Problem 1

The results of the uninformed search strategies performance on Problem 1 are summarized in 1.

Optimality The breadth-frst and uniform-cost search strategies were able to find the optimal plan for the Problem 1. The depth-first graph search and depth-limited search found plans that are not optimal. The results are in accordance with the theoretical performance of the search strategies as depth-first search strategies are not optimal.

Time complexity All of the search strategies found the solution to Problem 1 within 0.1 second.

Space complexity The depth-first graph search had the best space complexity performance, while the depth-limited search performed the worst.

Table 1: Comparison of the uninformed search strategies performance on Problem 1

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Breadth-First Search	6	43	56	180	0.03
Uniform-Cost Search	6	55	57	224	0.04
Depth-First Graph Search	12	12	13	48	0.01
Depth-Limited Search	50	101	271	414	0.09

2.3 Problem 2

TThe results of the uninformed search strategies performance on Problem 2 are summarized in 2.

Optimality Again, the breadth-first search and uniform-cost search strategies found the optimal plans, while the depth-first graph search and depth-limited search found plans that are not optimal.

Time complexity Most of the search strategies manage to find the solution to Problem 1 within 14 seconds, however the depth-limited search returned the result only after 1561 seconds. One surprising result is that the uniform-cost search performed better than the breadth-first search - in theory it should be the other way around. It is suspected that in this case this is caused by the code implementation of these two algorithms.

Space complexity The depth-first graph search has the best space complexity while the depth-limited search performs the worst. Uniform-cost search strategy has worse space complexity than the breadth-first search strategy.

Table 2: Comparison of the uninformed search strategies performance on Problem 2

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Breadth-First Search	9	3343	4609	30509	13.12
Uniform-Cost Search	9	4852	4854	44030	12.65
Depth-First Graph Search	1444	1669	1670	14863	13.21
Depth-Limited Search	50	222719	2053741	2054119	1561.31

2.4 Problem 3

The results of the uninformed search strategies performance on Problem 3 are summarized in 3. The depth-limited search was unable to finish running within a reasonable time-frame (1 hour).

Optimality Again, only two search strategies - the breadth-first search and uniform-cost search - found the optimal plan for Problem 3.

Time complexity The depth-first graph search was the fastest algorithm, while the breadth-first search was the slowest.

Space complexity The depth-first graph search has the best space complexity while the uniform-cost search perfomed the worst.

Table 3: Comparison of the uninformed search strategies performance on Problem 3

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Breadth-First Search	12	14663	18098	129631	150.25
Uniform-Cost Search	12	18235	18237	159716	104.76
Depth-First Graph Search	571	592	593	4927	4.04
Depth-Limited Search	NA	NA	NA	NA	NA

3 Heuristic search result strategies

In this section these four heuristic (informed) search strategies are compared in their ability to solve problems 1 - 3:

1. Greedy Best-First Graph Search
2. Recursive Best-First Search
3. A* Search Ignore Preconditions
4. A* Search Level Sum

Greedy best-first graph search is an informed search strategy, which expands the node closest to the goal. This strategy is not **optimal** (Russell and Norvig 2010).

Recursive best-first search is an algorithm similar to recursive depth-first search, uses linear space and tries to mimic the operation of the standard best-first search. It is **optimal** and can solve problems, which are not solvable using A* due to memory limitations (Russell and Norvig 2010).

A* search with ignore preconditions heuristic is an A* search strategy, which uses a heuristic funtion h, which returns the number of unsatisfied goals.

A* search with level sum heuristic is an A* search strategy, which uses a heuristic funtion h, which returns the sum of the levels in which the goals are in.

3.1 Problem 1

The results of the heuristic search strategies performance on Problem 1 are summarized in 4.

Optimality All of the heuristic search strategies found the optimal plan for the Problem 1.

Time complexity The recursive best-first search was the slowest (2.53 seconds) of the four algorithms, while greedy best-first graph search found the solution in 0.01 second.

Space complexity The greedy best-first graph search is the most memory efficient and the recursive best-first search had the worst space complexity. The difference in the results is huge the greedy best-first graph search expanded 600 times fewer nodes than the recursive best-first search.

Table 4: Comparison of the heuristic search strategy performance on Problem 1

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Greedy Best-First Graph Search	6	7	9	28	0.01
Recursive Best-First Search	6	4229	4230	17029	2.53
A* Search Ignore Preconditions	6	41	43	170	0.04
A* Search Level Sum	6	11	13	50	0.64

3.2 Problem 2

The results of the heuristic search strategies performance on Problem 2 are summarized in 5. The recursive best-first search was unable to finish running within a reasonable time-frame (1 hour).

Optimality A* search with ignore preconditions heuristic and A* search with level sum heuristic found the optimal plan while the greedy best-first graph search did not.

Time complexity Again, the greedy best-first graph search was the fastest algorithm. The A* search with ignore preconditions heuristic was about 2 times slower and the A* search with level sum heuristic was around 35 times slower.

Space complexity In the case of Problem 2 the most memory efficient strategy is the A* search with level sum heuristic. It expanded around 11 times fewer nodes than the greedy best-first graph search and around 17 times fewer nodes than the A* search with ignore preconditions heuristic.

Table 5: Comparison of the heuristic search strategy performance on Problem 2

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Greedy Best-First Graph Search	21	990	992	8910	2.52
Recursive Best-First Search	NA	NA	NA	NA	NA
A* Search Ignore Preconditions	9	1450	1452	13303	5.07
A* Search Level Sum	9	86	88	841	85.84

3.3 Problem 3

The results of the heuristic search strategies performance on Problem 2 are summarized in 6. The recursive best-first search was unable to finish running within a reasonable time-frame (1 hour).

Optimality Again, the A* search with ignore preconditions heuristic and A* search with level sum heuristic found the optimal plan while the greedy best-first graph search did not.

Time complexity Again, the greedy best-first graph search was the fastest algorithm. The A* search with ignore preconditions heuristic was about 1.2 times slower and the A* search with level sum heuristic was around 17 times slower. The difference in the results in this problem is significantly smaller than in Problem 2.

Space complexity In this problem the most memory efficient strategy is again the A* search with level sum heuristic. It expanded around 18 times fewer nodes than the greedy best-first graph search and around 16 times fewer nodes than the A* search with ignore preconditions heuristic.

Table 6: Comparison of the heuristic search strategy performance on Problem 3

Search Strategy	Plan Length	Expansions	Goal Tests	New Nodes	Time
Greedy Best-First Graph Search	22	5614	5616	49429	34.95
Recursive Best-First Search	NA	NA	NA	NA	NA
A* Search Ignore Preconditions	12	5040	5042	44944	41.97
A* Search Level Sum	12	318	320	2934	594.95

4 Comparison of results for all search result strategies

After analysing the performance of the informed and uninformed search strategies it is clear that most of them have their advantages and disadvantages. Some strategies are optimal, but are slow and not memory efficient, while others are not optimal but have geat memory efficiency or are very fast. In the following subsections the search strategies are compared taking one of the criteria (e.g. optimal) as the primary factor and then comparing the search strategies on the other criteria. This way it is easy to determine which search strategy ought to be used in different cases - if optimality is essential then which of the optimal search strategies have the best memory and speed complexity. On the other hand if the memory is the limiting factor, which of the search strategies can find the optimal solution and which of them can find any solution in the shortest amount of time.

4.1 Optimality

The optimality results are in accordance with the theory - the algorithms based on breadth-first search (breadth-first search, uniform-cost search and A* search) are guaranteed to find the optimal plan. The depth-first search and best-first search strategies sometimes managed to find the optimal plans, but often did not. Thus, if optimality is essential the breadth-first search strategies are the best choice. Not surprisingly the heuristic algorithms outperformed the uninformed search strategies.

Time complexity The A* search with ignore preconditions heuristic is the fastest of the optimal search strategies. For the analysed problems it is 2-3 times faster than the uninformed breadth-first and uniform-cost search strategies. A* search with level sum heuristic is the slowest of the optimal search strategies and can be more than 10 times slower than the A* search with ignore preconditions heuristic for the analysed problems.

Memory complexity The A* search with level sum heuristic is by far the most memory efficient optimal search strategy. It expanded more than 10 times fewer nodes than the A* search with ignore

preconditions heuristic, around 45 times fewer nodes than the breadth-first search and 55 times fewer nodes than the uniform-cost search.

Overall performance The A* search with ignore preconditions heuristic is probably the best optimal search strategy for most cases - it is the fastest and the second most memory efficient. In cases where memory efficiency is critical and the speed is less important, the A* search with level sum heuristic should be used. The uninformed search strategies perform significantly worse.

4.2 Time complexity

The time complexity for the search strategies vary widely. Solving just 3 problems does not give nearly enought information to make general conclusions about the speed of the various search strategies, however some search strategies concistently perfomed better than others. This time it is not clear if the informed search strategies have advantage over the uninformed strategies - the heuristic greedy best-first graph search was fastest in the Problem 2, but the uninformed depth-first graph search was the fastest in Problem 3, while Problem 1 is not usefull for determining the time complexity since most of the strategies found the solution in around 0.01 seconds. The problem is with the results of the Problem 3 - the depth-first graph search found a very bad plan with a length of 571, but managed to do it very fast and without expanding many nodes.

Optimality The two fastest search strategies - greedy best-first graph search and depth-first graph search are not optimal. However, the A* search with ignore preconditions heuristic is quite fast too - in Problem 2 it outperformed the depth-first graph search and in Problem 3 it was just 1.2 times slower than the greedy best-first graph search.

Memory complexity The memory efficiency of the algorithms is not clear. Again, the two fastest strategies are not very memory efficient and it is not clear which one is the most memory efficient. The greedy best-first graph search is quite memory efficient and A* search with ignore preconditions heuristic is quite similar.

Overall performance The greedy best-first graph search is probably the fastest of the algorithms with the A* search with ignore preconditions heuristic performing similarly but a bit worse. On the other hand the A* search with ignore preconditions heuristic is optimal. It is hard to determine the true performance of the depth-first graph search - it is very fast in Problem 3 and not impressive in Problem 2.

4.3 Memory complexity

The A* search with level sum heuristic is clearly the best search strategy if memory complexity is an issue. It is many times more memory efficient than other strategies.

Optimality The A* search with level sum heuristic is an optimal strategy, therefore it is clearly the best choice if memory complexity and optimality is required.

Time complexity The A* search with level sum heuristic is one of the slowest search strategies considerably slower than other optimal or fast strategies. There is no strategy which would be simultaneously fast and memory efficient. The A* search with ignore preconditions heuristic has the best tradeoffs - it is one of the fastest and one of the most memory efficient strategies.

Overall performance Clearly the A* search with level sum heuristic is the best strategy if memory efficiency is the most important factor. The A* search with ignore preconditions heuristic is a good choice if the search strategy also has to be fast. As an added bonus both of these strategies are optimal.

5 Conclusion and recommendations

The detailed analysis of the search strategies showed that the heuristic search strategies have a clear advantage. Based on the analysis done in this paper A* search with ignore preconditions heuristic is determined to be the best search strategy for all cases, except when memory efficiency is very important, where A* level sum search is the best.

References

Russell, Staurt J and Peter Norvig (2010). *Artificial Intelligence (A Modern Approach)*.