

Design A* Algorithm for Sokoban Game: Heuristics is All You Need!

Quan Hoang Ngoc - 22521178
3/2024

In this project, we install an A search algorithm to apply it to solve the Sokoban game. We design a new heuristic function for the A* algorithm and make a benchmark to evaluate its performance with the default heuristic function as well as the UCS algorithm.*

A. Theoretical Background

- To understand the theories presented below, people need knowledge about some concepts such as Search Problems, Modeling Search Problems, Search Algorithms, and Heuristics Functions.

1. Search Algorithm: UCS, Greedy, and A*

- UCS is an algorithm whose strategy is to expand the lowest path cost. It is a *complete and optimal* search algorithm. But, the bad of the algorithm is it explores options in *every "direction"* because it has no information about goal location.
- Greedy overcomes the weakness above because its strategy is to expand a node that it thinks is closest to a goal state. It performs this by an *heuristic function*. Thus, it frequently finds a solution quickly but does not ensure that it is an optimal solution.
- The perfect combination of the two above algorithms is the A* algorithm when it uses both information about backward cost (path cost) and estimated forward cost (heuristics). In other words, the strategy of A* is to expand the node with the lowest sum of path cost and estimated cost. Now, let us present that in formula form:
 - For each node n : $f(n) = g(n) + h(n)$
 - $g(n)$: backward cost (path cost)
 - $h(n)$: estimated forward cost - It is a heuristics function

2. Heuristics Function: there is a tradeoff between accuracy and speed

- Pay attention to the above formula and we will present a few comments below:
 - **Heuristic design is a key**: In the A* algorithm mentioned above, designing a good heuristic function is almost the majority of the algorithm's work.
 - Unfortunately, we assume that there is a *trade-off* between the accuracy (optimality) and the speed of the algorithm.
- Let's try to simplify the formula and consider the $h(n)$ function as a hyperparameter H , then $f(n) = g(n) + H$.
 - If $H \rightarrow 0$, then A* will tend to take the form of UCS and have its properties. Thus, the solution that is found will be quite near an optimal solution.
 - But, if $H \gg 0$, then the weight of H will overwhelm the $g(n)$ value and it tends to take the form of Greedy. So, It may quickly find a solution, but it is not necessarily optimal. It will be not optimal in some cases leading to the heuristic function not admissible (or consistent).

B. Heuristic Ideas Explanation

- Typically, we will design a heuristic function by solving a more relaxed problem than the original problem.

1. Default Heuristic Idea

- Imagine that we play the Sokoban Game on a flat and spacious map, then the minimum cost of pushing a box to goal position will be equal to their Manhattan distance.
- Using that idea we will present it in formula form as follows:
 - Index for N boxes that do not correct position and goal positions.
 - $(p1, p2, \dots, pN)$ is a permutation of N .
 - Then, the heuristic for a state is the *sum of the Manhattan distance between (i-th box Position, p[i]-th goal Position)*.

2. Q Heuristic Idea

- Through running experiments, we found that there will be some maps where the default function $h(n)$ will not find an optimal solution.
- We hypothesize that the function $h(n)$ above is larger than allowed and we will refine to get a smaller heuristic function in the hope of finding optimal solutions with not significantly more time.
- We use Euclidean distance instead of Manhattan distance. Besides, we choose the permutation intentionally instead of randomly.
 - $(p1 = j1)$, which is an index of the goal position closest to the 1-th box
 - $(p2 = j2, j2 \neq p1)$, which is an index of the goal position closest to the 2-th box, etc
 - Then, the heuristic for a state is the *sum of Euclidean distance between (i-th box Position, p[i]-th goal Position)*.

C. Experiment and Evaluation

- To evaluate the performance of algorithms, we conducted experiments to measure some quantity such as: the length of solution (number of actions that need to go to goal state) - \underline{L} , and the number of nodes (state) that expended by algorithm - \underline{N} , and the run time of algorithm - $\underline{T}(s)$.

Statistics Table-[1]

	UCS		A* (default)		Q* (Q heuristic)	
Level	N	T	N	T	N	T
1	818	0.26	192	0.06	213	0.11
2	82	0.02	51	0.01	60	0.02
3	677	0.35	75	0.04	175	0.12
4	89	0.01	54	0.01	54	0.02

5	687417	225	1010	0.45	3234	1.71
6	250	0.03	221	0.03	231	0.05
7	7110	1.32	1095	0.20	2005	0.45
8	2425	0.40	2425	0.73	2434	0.46
9	105	0.03	68	0.02	44	0.01
10	230	0.09	203	0.03	203	0.04
11	300	0.04	289	0.04	294	0.04
12	1266	0.18	684	0.12	771	0.14
13	2382	0.28	1928	0.59	2117	0.38
14	27440	5.86	10688	2.95	17044	5.27
15	2519	0.51	2241	0.71	2356	0.92
16	70366	31.56	1603	0.81	13372	7.87
17	71595	36.73	71595	57.54	71595	45.95
18	OOM		OOM		OOM	
	867731	302.67	94230	64.34	116202	63.56

Notion:

[x]: not find a solution

[OOM]: Out of Memory

The computer used for statistics is ASUS Vivobook: Core Intel i5, 12 GB RAM.

1. Evaluate Optimality (number of actions)

- To evaluate the optimality of algorithms, we will measure the length of solution that can be found by each algorithm.

Length and Optimality Table-[2]

	Length			Optimal		
Level	UCS	A*	Q*	UCS	A*	Q*
1	12	13	12	True	False	True

2	9	9	9	True	True	True
3	15	15	15	True	True	True
4	7	7	7	True	True	True
5	20	22	20	True	False	True
6	19	19	19	True	True	True
7	21	21	21	True	True	True
8	97	97	97	True	True	True
9	8	8	8	True	True	True
10	33	33	33	True	True	True
11	34	34	34	True	True	True
12	23	23	23	True	True	True
13	31	31	31	True	True	True
14	23	23	23	True	True	True
15	105	105	105	True	True	True
16	34	42	34	True	False	True
17	x	x	x	True	True	True
18	OOM	OOM	OOM			

- We know that UCS always finds an optimal solution, so if A* and Q* find a solution whose length is equal to the solution found by UCS, it will be an optimal solution.
- Following the Table-[2] above, **A* is not to find an optimal solution** in maps including **level 1, level 5, and level 16** whose maps do not contain walls.
- In particular, **Q* always finds an optimal solution** in all maps.


2. Evaluate Run Time

- Following Table-[1], in most maps, A* frequently finds a solution that is faster than Q*. The proof of this is the sum of nodes of A* is smaller than Q*.
- But, in some maps that need more time to solve, Q* finds a solution faster and more correct than A*. The result is the **sum of the run-time of Q* is a little better than A***.

D. Conclusion

- The most important thing in the A* algorithm is designed to be a good heuristic function.
- There is a tradeoff between optimality and speed when designing heuristic functions.
- We also presented various ideas for the design heuristic function, where the heuristic function Q^* that we proposed can find optimal solutions in all maps while the search time is not significant.

E. References

- All about source code, experiment process documentation, and Youtube demo (if there) in this Github link: https://github.com/QuanHoangNgoc/Astar_Heuristic_for_Sokoban_Game
- [Experiment_docs.pdf](#)
-  A* and Sokoban Game - Heuristic is piece of cake! - [AI course] (Please Like and Subscribe)
- Author: Quan Hoang Ngoc - 22521178. Thank you.