KI2: Künstliche Intelligenz - SS22
AI Project: Rush Hour 1

Samuel Ebner, Lisa Lamplmair, Sebastian Mayer

May 10, 2022

Abstract

In this programming assignment, we were tasked with using the A* algorithm to solve instances of the Rush Hour puzzle. This involved implementing a graph-search version of A*, along with three heuristics. Our implementation was tested on several Rush Hour puzzles.

1.1 Algorithm

For finding the best possible solution for the Rush Hour Puzzle the A* Algorithm is used in this project. With a weighted graph of search nodes, starting from one specific point, it finds a path to the goal with the smallest costs, which would be the exit of the red car in this case. The A* algorithm uses an open and closed list to ensure, that nodes are not visited twice. For each step the algorithm has to determine which node to expand next. This is evaluated with the lowest overall costs, this is done by minimizing $f(n) = g(n) + h(n)$. In this project the cost for one move is always 1, so the path costs are equal to the depth of the node. The heuristic is calculated for each node individually. In this project for sorting the nodes according to their overall costs a PriorityQueue was used. This Queue automatically sorts all inserted nodes according to a specific comparator. In the closed list every node which is already expanded is stored. In this list every node can only be once, so a Set was used to implement that.

1.2 Heuristics

The heuristics should analyse the current state of the puzzle and make an estimation of how many moves would be necessary when visiting the current node. For the heuristics it is important to never overestimate the real costs (moves needed). As a result the heuristic would not be admissible and therefore the solution found may not be optimal.

1.2.1 Zero-Heuristic

The project already implemented a Zero Heuristic, which returns zero for each node. Using A* with this heuristic is equivalent to breadth-first search.

1.2.2 Distance Heuristic

We also considered the distance from the goal car the exit as a possible heuristic (the results are also included below). When investigate this heuristic one can see that this would not be admissible for all possible puzzles. Because one car can move as far as possible in one move, the distance could possibly overestimate the real costs and not find an optimal solution. Therefore this heuristic is not admissable. See also Chapter 1.3

1.2.3 Blocking Heuristic

As second heuristic a blocking heuristic was considered. It counts all others cars, blocking the exit for the red car. Each car counts one, because at least one move would be required to get is out of the way. This value plus one for the goal car (to move the goal car to the exit, one move is required) is the result for a state.

1.2.4 Advanced Heuristic

In the advanced heuristic the blocking heuristic was extended, to not only count the cars blocking the goal car, but also the cars which block these cars again, and the cars which again block these, and so forth. The idea behind this algorithm was to count all blocking cars in a recursive way.

For considering the blocking cars in a recursive way, much information is needed to accomplish a correct value. First we counted all blocking cars of our goal car again and did the recursive call only for those cars. Each car had to be tested for moving backwards and forwards and then the overall count of which is the lowest of both was chosen. This was done because there could be more possible solutions and we want the minimum required moves. Next we had to determine whether a car has enough space to move out of the way, is blocked by another car or by a wall of the game. When a blocking car is found the recursive call is done. In order to not produce an endless recursion a list with all cars already considered was taken into account. Although this means that cars can be never counted twice, so if one car blocks three others it is only counted once and then considered to have been moved out of the way. This is also necessary to not overestimate the real costs.

We also tried the add the distance to the Value calculated for the Advanced Heuristic, but this lead of course to the same problem as mentioned beforhand in Chapter 1.2.2.

1.3 Consistent and Admissible

Both Blocking and our Advanced Heuristic are consistent and therefore also admissible. A heuristic is consistent if its estimate is always less than or equal to the estimated distance from any neighbouring vertex to the goal, plus the cost of reaching that neighbour. A heuristic is admissible if it never overestimates the cost of reaching the goal, for example the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.

As we count each car once, and it is counted to have at least one move to be moved out of the way of our goal car, the estimated cost is always smaller or equal to the actual cost as it has to be moved at least once. Here the length of the move is not of importance.

1.4 Results

In the following table the results for all five heuristics can be seen (Zero - Blocking - Distance - Advanced - AdvancedDistance). Note that the Heuristics using the Distance are performing quite good but are not admissible. It can be seen that some results get much worse results comparing to the admissible heuristics, because it did not find the optimal solution. We accomplished an admissible heuristic with our Advanced Heuristic which performs considerable better than the blocking heuristic. When running the project it can clearly be seen, that the more difficult the puzzle gets, the longer it takes the algorithm to find a possible solution because many recursive calls are needed for all possible states of the puzzle. In this matter the blocking heuristic is the one which obviously performs better.

All in all our heuristics performed as expected. The only surprising thing was to see that for some individual puzzles some 'lesser' heuristics performed better or just as good, but overall worse. As the puzzles are listed roughly in order of difficulty it was interesting to see how the algorithm behaved. The more cars clustered in one spot, the harder it was for the algorithm to get to a solution whereas for the human eye clusters were detected easily and resolved with more ease. In comparison, the more dependencies on blocking cars were found, the harder it was for the human eye and mind to have an overview over the situation as only a few moves could be thought through before moving.

	Zero			Blocking			Distance			Advanced			AdvancedDistance		
name	nodes	dpth	br.fac	nodes	dpth	br.fac	nodes	dpth	br.fac	nodes	dpth	br.fac	nodes	dpth	br.fac
Jam-1	11589	8	3,066	6829	8	2,857	9561	9	2,625	991	8	2,196	959	8	2,186
Jam-2	24081	8	3,378	4044	8	2,663	11407	9	2,681	2643	9	2,248	1570	8	2,340
Jam-3	7731	14	1,788	4059	14	1,699	4249	15	1,639	3688	14	1,686	3410	15	1,613
Jam-4	3203	9	2,301	1281	9	2,057	2707	9	2,255	409	9	1,781	206	9	1,627
Jam-5	21390	9	2,888	5075	9	2,433	15724	11	2,284	668	9	1,896	398	9	1,774
Jam-6	15992	9	2,791	6567	9	2,510	12786	11	2,239	2287	10	2,025	2465	11	1,901
Jam-7	52493	13	2,202	20143	13	2,035	24401	14	1,955	19262	13	2,027	18272	13	2,018
Jam-8	6461	12	1,957	5381	12	1,925	6064	13	1,840	972	12	1,641	212	12	1,411
Jam-9	6116	12	1,947	3200	12	1,835	3269	12	1,839	2468	12	1,791	1549	12	1,715
Jam-10	15890	17	1,675	12739	17	1,651	13721	18	1,608	8001	17	1,602	8186	18	1,558
Jam-11	6694	25	1,347	5778	25	1,338	6017	26	1,324	4961	25	1,329	4554	25	1,324
Jam-12	11677	17	1,642	5457	17	1,562	5541	17	1,564	4656	17	1,546	3024	17	1,502
Jam-13	69130	16	1,916	31112	16	1,816	62786	18	1,763	19842	16	1,761	22037	19	1,608
Jam-14	97411	17	1,880	39873	17	1,776	59925	18	1,758	44531	18	1,727	60397	18	1,759
Jam-15	3180	23	1,338	3171	23	1,337	3175	23	1,337	3171	23	1,337	2869	23	1,331
Jam-16	21560	21	1,529	17597	21	1,513	16791	21	1,509	16879	21	1,510	15631	22	1,473
Jam-17	19560	24	1,436	16575	24	1,425	18025	25	1,408	10451	25	1,375	7752	24	1,376
Jam-18	13839	25	1,392	10540	25	1,375	12179	25	1,384	7202	25	1,352	6934	25	1,350
Jam-19	3610	22	1,367	3463	22	1,364	3453	22	1,364	3482	22	1,364	3224	22	1,359
Jam-20	12593	10	2,438	3116	10	2,095	7171	11	2,115	1973	10	1,992	3926	11	1,992
Jam-21	1650	21	1,332	1535	21	1,327	1631	21	1,331	1301	21	1,315	1162	21	1,306
Jam-22	31244	26	1,421	22294	26	1,401	25421	26	1,408	11322	27	1,343	10358	27	1,338
Jam-23	18382	29	1,338	10343	29	1,309	11546	30	1,301	10163	29	1,308	9354	33	1,257
Jam-24	45187	25	1,467	42796	25	1,463	43034	25	1,463	42408	25	1,463	40432	25	1,459
Jam-25	80491	27	1,455	61820	27	1,440	78720	28	1,433	54025	27	1,432	55422	28	1,414
Jam-26	40135	28	1,396	33715	28	1,386	35803	29	1,372	29348	28	1,379	26062	28	1,372
Jam-27	20228	28	1,359	17448	28	1,351	18224	29	1,338	15462	28	1,344	13807	29	1,323
Jam-28	14809	30	1,313	9813	30	1,293	10859	31	1,286	4299	30	1,253	2203	30	1,221
Jam-29	38385	31	1,345	37766	31	1,345	38130	32	1,331	34569	31	1,340	31109	31	1,335
Jam-30	8580	32	1,264	7675	32	1,259	8014	33	1,251	7060	32	1,255	6599	33	1,242
Jam-31	32409	37	1,270	30623	37	1,268	30663	39	1,251	28863	37	1,265	28248	38	1,256
Jam-32	3155	37	1,182	2800	37	1,178	3175	40	1,165	2683	37	1,176	2674	38	1,170
Jam-33	36459	40	1,249	23243	40	1,233	27331	41	1,232	15743	41	1,213	13938	41	1,209
Jam-34	37447	43	1,228	34899	43	1,226	36668	44	1,221	31437	44	1,217	29199	44	1,214
Jam-35	33989	43	1,225	32919	43	1,224	33469	43	1,225	32553	44	1,218	29998	44	1,215
Jam-36	22309	44	1,206	17182	44	1,198	18339	45	1,195	15739	44	1,195	15613	45	1,190
Jam-37	15268	47	1,179	14245	47	1,177	15154	48	1,174	8919	47	1,164	5827	47	1,152
Jam-38	28374	48	1,192	23526	48	1,187	24530	49	1,183	22597	48	1,186	22723	49	1,181
Jam-39	24708	50	1,179	24236	50	1,178	24878	53	1,167	23908	50	1,178	23861	50	1,178
Jam-40	24469	51	1,174	20347	51	1,170	22435	51	1,172	17615	51	1,166	14991	51	1,162

Table 1.1: Comparison of Heuristics. Zero - Blocking - Distance - Advanced - AdvancedDistance