Mini-Project 2 - Analysis

Rush Hour

A Report
Presented to

The Department of Electrical & Computer Engineering

Concordia University

In Partial Fulfillment
of the Requirements
of COMP472

By:
Team Artificial Unintelligence

Paul Westenberg *ID: 40087074*
Saro Nokhoudian *ID: 40073669*
Nicolas Bernier-Nguyen *ID: 40135649*

Presented to:
Dr. Leila Kosseim
Sunday, November 26th 2022

Question 1 :

The length of the solutions across algorithms and heuristics. When do you have the lowest-cost solution?

Solution:

In order to explain each algorithm as well as its advantages, disadvantages, features…, we will use the solution for puzzle 2 of the input.txt file. This was generated by running our code on Jupyter Notebook, generating the excel file with all features. We will use the following table in order to explain why certain algorithms performed the way they did. Please note that we are only looking at 1 puzzle in the analysis as analyzing all 6 of the input files would be tedious. However the rest can be found in our solution where each puzzle's solution is found in a separate folder to be consulted.

Table 1: Puzzle 2 results across all algorithms from input file

Algorithm	Heuristic	Length of Solution	Length of Search Path	Execution Time
UCS	NA	10	1836	15.90579
GBFS	h1	10	113	0.864389
GBFS	h2	10	113	0.790474
GBFS	h3	10	113	0.787652
GBFS	h4	15	174	1.230057
A / A*	h1	10	297	2.672071
A / A*	h2	10	297	2.823723
A / A*	h3	10	150	1.103575
A / A*	h4	13	148	0.867771

UCS: The **Uniform Cost Search** algorithm goes through all the node possibilities depending on the value of f(n) or g(n) (as they are the same) until it finds a node where the AA car is in the correct position (at the end of its row). UCS does not have any heuristic values but rather relies on the current move count (g(n)) to determine which node it will be running. In theory, UCS should always yield the best possible move cost as it verifies all the possibilities of nodes. This of course has its drawbacks such as time which we discuss in question 3.

From our results we observe exactly what was expected. For each of the puzzles, the UCS algorithm finds the lowest cost solution. For example, in puzzle 2 (table 1), the lowest cost solution is 10 moves and UCS successfully finds it in 10 moves. This happens for all puzzles (50 random puzzles tested) as it finds the lowest cost solution without fail. We can also observe that the length of the search path is much larger than all of the other algorithms. This is normal as UCS analyzes all the

possible nodes for each possible car move of a particular position resulting in a large number of visited nodes (positions).

GBFS: The **Greedy Best-First Search** algorithm is an informed search algorithm which considers only h(n) when making the decision on which nodes to explore next. GBFS keeps track of g(n) as the number of moves the current node has come from the initial state but doesn't have f(n). It uses only h(n), the cost indicated by h which is the prediction of total moves needed to reach the solution. This heuristic value h(n) is found depending on which h1 we are choosing. More on this in question 2 where we analyze each heuristic counter separately. However, GBFS does not always find the lowest cost solution as once it takes a particular path, it will always observe the nodes with the lowest h first which don't necessarily guarantee the lowest cost solution. Furthermore, once one of its nodes finds a solution (that is when car A reaches the end of row 2), the algorithm stops giving the results. It doesn't try to find a better solution but rather takes the quickest one found. This can be seen in table 1 where we analyze puzzle 2, as we can see that although the cost is not optimal, the time is much faster than its peer UCS. These results are similar across all the puzzles. Therefore, GBFS will execute faster than other algorithms, but it cannot guarantee a lowest solution path. We will expand more on the search path length as well as the time in question 3.

A / A*: Please note that since the assignment asked us to do A or A*, we decided to go with algorithm A. The **A** algorithm is an informed search algorithm. The formula of algorithm A in order to prioritize certain nodes above others is given by f(n) where f(n) = g(n) + h(n). Here, g(n) is the current cost of the path till the present node n, and h(n) is the cost indicated by the heuristic value which is the prediction of the number of moves to reach the solution. f(n) is the estimated total cost of the solution for each node. So the algorithm will always execute the nodes with the lowest f(n) until it reaches the solution. Since Algorithm A considers both h(n) and g(n) which means it keeps track of each path taken to be able to come back to it if other paths are getting higher costs of f(n). Therefore, algorithm A should be able to give a low solution path but necessary as low as UCS, but it will execute much faster then UCS. On the other hand, A* would have been the best algorithm since it guarantees the lowest path and executes faster then UCS.

To conclude on our findings of lowest cost solution across the various algorithms, we can observe that UCS has the lowest-cost solution as visible in table 1. This is true across all the puzzles of our input file. A close second is A, where we observe that for h1, h2 and h3 it yields the lowest cost solution (10 in puzzle 2) but not for h4 (13). This is because we decided to go with A and not A*. From theory, we know that UCS as well as A* both yield the lowest cost solution. We thus know that had we implemented A* and not A we would have also obtained the lowest possible cost for each puzzle for A*. Lastly, after going through our results, GBFS doesn't focus on obtaining the lowest possible solution cost but rather on the speed at which it finds an answer. More on this in question 3 where we analyze the speed and search path lengths of our algorithms.

Question 2:

The admissibility of each heuristic and its influence on the optimality of the solution.

Solution :

For **h1** (the number of blocking vehicles):

The **h1** heuristic value will calculate the h(n) by counting the number of cars to the right of the car A. At first, this is a logical heuristic value as an approximation of the number of moves needed to complete the puzzle as car A needs to go out of the puzzle from the right. It is thus more logical to observe hodes where h1=2 rather than h1=4 as the former is more likely to obtain the solution faster since we only need to move 2 cars out of the way and not 4. The issue with h1 is that usually it will give a value between 0 to 4, which means the estimated cost of the number of moves needed to complete the puzzle. However, as an example, if we take table 1, that is puzzle 2's starting node, in reality it will need 10 moves to complete the puzzle whereas the initial heuristic value of the first observe node (the starting position) will be h1(n) = 2 as there are only 2 cars (K and L) blocking car A from advancing. It is thus not accurate in predicting how many more moves until the solution is reached. That being said, even though it does not always give the best path to finding the solution, as far as estimations go, it is a logical one as for A to go forward, the cars in front of A need to move and so going through the nodes with the lowest cars in front of A will lead to a quicker solution. This can be seen across our 50 analyzed random puzzle excel sheet results contained in the zip file.

For **h2** (the number of blocked positions):

The **h2** heuristic value will calculate h(n) depending on the number of blocked positions to the right of car A. We can observe that in the case where the cars blocking A are all vertical, then we will have the exact same heuristic value as h1. The only way h2 will defer from h1 is if there is a horizontal car blocking A. In that case, the number of vehicles blocking will be 1 whereas the number of blocked positions will be 2 or 3 (depending on the size of the vehicle). In the input file given, none of the problems have this scenario where a horizontal car is blocking A and thus, for all of the problems in our input file, all h2 results will be exactly the same as h1. We thus included in our input file an example of this and indeed h1 and h2 yielded different results. If there is only one horizontal car blocking car A then h2 will be better than h1. The reason of this is, if there is a horizontal blocking the right side of car A h1 will predict h(n) of 1 but in reality we will need two moves (1 to bring the blocking car out of puzzle and 1 to move A to the right), h2 will give h(n) more than 1 and that will be better prediction of reality. Another case will be if there are horizontal and vertical cars blocking A; then h2 will be slightly better because most of the time h2 will give higher h(n) because of the number of positions blocked and reality moves would be higher too. These thus depend on the particular scenario of the puzzle. In a similar way to h1, h2 is fairly efficient in prioritizing certain nodes in the open list as moving cars out of A's way is efficient. h2 thus results in an efficient focus on particular 'more likely to find solution' nodes, ultimately not looking

at nodes with larger heuristic values as this would mean that there are more positions blocking A. We thus save a lot of time (as will be seen in question 3) as our search path length is optimized. However this comes at the cost of our solution cost which we have seen in question 1.

For **h3** (value of h1 multiplied by 3):

The **h3** heuristic value is found by taking h1 and multiplying it by lambda which we have decided to be 3. When we use h3 in GBFS we will have the same exact output as h1 since GBFS will only consider h(n) and since every h(n) is being multiplied with a constant value then mathematically we can eliminate that constant value and end up with exact same values of h1(n). For example, in puzzle 2, at the starting node, h1(n) = 2 meaning h3(n) = 2*3 = 6. Repeating this for all nodes and going with the lowest node each time will yield the exact same results as we have simply multiplied by 3 all heuristic values. It will thus, at each instance, go with the exact same node as it did in h1, resulting in exactly the same findings for h1 and h3 in GBFS. However, that is not the case with algorithm A. Algorithm A will also consider g(n) and that will change the results. This is because in A, we consider both g(n) and h(n) in making our decision about which node to explore from the open list. This lambda value of 3 thus puts a bigger weight (3x) on the heuristic value rather than g(n) meaning the decision is much more influenced by what our estimate on the number of moves until the end (h3(n)) rather than g(n). The higher the lambda value we chose the higher we are prioritizing and the more weight we are giving to h(n) rather than g(n) in making the decision on which node to explore next.

For **h4**:

The **h4** heuristic is calculated by summing all the blocked positions to the right side of car A on every single row. For instance, in puzzle 2, the number of blocked positions to the right of A on all rows is 7 as we have K,K,K,E,L,L and E. This means that h4(n) will yield a higher heuristic value compared to h1 and h2. We can see in our outputs that for a large number of cases of h4, it is providing a longer solution path for GBFS and algorithm A. This is because of the logic into the system that we are providing. The main reason for this behavior is that the algorithm will try moving the horizontal cars to the left of car A and after moving car A to the right minimizing h4. This happens often and is inefficient as we do not need to move the horizontal cars and waste moves on them. The reason for moving the horizontal cars first is caused by the values of h being predicted by every possible new move. When the algorithm is trying to find the best movement, it is also trying to move the horizontal cars to the left and it is seeing that they are providing lower value of h since it is having less position to the right side of car A. The payoff for not having the best cost solution is in the time it takes to calculate. As we can see from table 1, h4, although having more moves than its peers h1,h2 and h3, completes and finds its solution in a much faster time (0.8 compared to 2.6 seconds). It is thus an interesting heuristic as it lacks in optimizing the number of moves it excels in time taken.

Question 3 :

The execution time across algorithms and heuristics. Is an informed search always faster?

Solution:

UCS: The **Uniform Cost Search** algorithm will choose the parent with the lowest cost and execute all its possible moves, and it will repeat until it gets the solution. Since UDP executes a huge amount of nodes (all possible nodes) it will take a significant amount of time, but it will ensure that we are getting the lowest possible solution path. This can be seen across all our puzzles, as UCS always has the largest time taken. Visible in puzzle 2, (table 1) as UCS takes 15 seconds whereas GBFS and A take around 2 seconds.

GBFS: The **Greedy Best First Search** algorithm will only consider h(n) and it will not keep track of the ancestors. Therefore, it will not go back to execute other nodes and it will not execute nodes with high heuristic values. Therefore, GBFS will have the lowest execution time compared to all other algorithms. This can be seen in all our results for the 50 random puzzles as GBFS almost always has the quickest time to the solution. This is due to its lower search path length, as it does not need to examine all the nodes, but rather only the ones with the lowest heuristics. It is thus a quicker way to obtain a solution but lacks in finding the optimal one as we saw in question 1.

A: Algorithm **A** will consider g(n) and h(n) as a sum in f(n), meaning it won't execute all the nodes the way UCS does but rather only the ones who have the lowest f(n). However it has the ability to keep track of its ancestors and it will go back if the ancestor has a lower cost of f(n). Therefore, algorithm A is faster then UCS but slower then GBFS. We can see this through our 50 random puzzle excel document and it is visible in puzzle 2, as A took about 2 seconds, UCS took 15 and GBFS took 1 approximately. Even though it takes longer than GBFS, it is much more accurate in finding the optimal solution meaning it is overall better than GBFS (well depending on what you want)

To conclude on our findings on the execution time, GBFS is the fastest of all three but isn't able to always obtain the lowest cost solution. A is much faster than UCS as it does not have to visit nearly as many nodes (which leads to a lower search path). However, A also does not always yield the most optimal solution. On other hand A*, as we know from theory, delivers the lowest cost solution all while minimizing its search path length and execution time. Lastly, UCS has a large execution time but guarantees the lowest possible solution cost every time.

Question 4 :

Other interesting facts that you deem worthy of describing.

Solution:

A couple interesting facts our team deems worthy of describing would be:

- The different heuristic values we had thought of were choosing for h4. For instance, an idea we had was calculating all blocking cars in front of car AA plus the horizontal cars on the 1st and 6th row and which are to the right side of the car AA. The reason for this system is that usually all vertical cars that are blocking the way need to go either up or down and to do that we will need to move the horizontal cars. Therefore we think that h(n) will be closer to the actual cost. This was only one of our many ideas to find the heuristic values. At each node. Comparing them all would have been interesting to follow up on.

- As mentioned in the project handout, 50 puzzles were generated and tested on all three algorithms with all 4 heuristic values. As observed in the Excel file, UCS has a much longer execution time compared to GBFS and A. However, it always gives the lowest-cost solution. As for GBFS and A, they have a much lower execution time (nearly twice as fast). This comes at a disadvantage though as they very often do not provide the lowest-cost solutions. An interesting observation that was made is algorithm A, using heuristic values 1 and 2, always returns the lowest-cost solution.

- Concluding on the best algorithm cannot be done. What we mean by this is the algorithm you choose depends on the project you have and the requirements you give importance too. If you want a quick solution, then GBFS is best. If you want the lowest cost solution with 100% accuracy then UCS is the most efficient. This means that we can't single one algorithm down and say that it is the absolute best. If someone asks which algorithm should I implement for my project then one's response should be, well what do you deem is important (speed, accuracy…?). However, through our findings and class, after looking at the differences in algorithms and the benefits and flaws of each, for this particular problem and puzzles, A* is the most efficient, but again that is only for this particular set of puzzles.

- The importance of choosing a good heuristic finder. The heuristic value is what drives your algorithm in certain directions and why it stops it from looking at other nodes. It is thus important to choose a meaningful and significant heuristic value.

Having a bad heuristic value can make GBFS and A not work properly. It should be relevant to solving the problem (in this case getting car A to the end).

- A* not being looked at. We were asked to implement either A or A* in the project and we went with A. However from theory and class, we know that A* is faster than UCS and always yields the best possible solution. It would have thus been interesting to look into a way of implementing A* as well.