COMP3411 Assignment 2 - Jeremy Chen (z5016815)

Question 1

a)

Nodes Expanded Per Search Algorithm

	start10	start12	start20	start30	start40
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*	29	21	952	17297	112571

b)

From the table of results obtained, we notice that in terms of efficiency, the uniform cost search with Dijkstra's Algorithm is the worst as it expands the most number of nodes and eventually runs out of memory at start12 and beyond. While the Uniform Cost Search is effective at searching for the optimal route in solving the puzzle, it must expand a large number of nodes to do so, as well as keep track of nearly all the nodes it has traversed (unless a shorter path to that node is found), thus making this search method very tedious and memory inefficient.

In comparison, the Iterative Deepening Search did not encounter memory issues since it only stores memory for the existing search path, which is then discarded after it has moved onto the next path. This is unlike that of UCS, which will store the memory of most of the paths it finds, unless it there is a more optimal one. However, while IDS is memory efficient, it requires expansion of a significant number of nodes and becomes time inefficient for more complex permutations of the puzzle. As shown in the table of results above, the time taken to expand the nodes necessary to solve the puzzle has increased exponentially, and at start30 and beyond, this search can no longer be done in a reasonable time period.

For A* Search, as shown in the table, there is a significant reduction in the number of nodes expanded, in comparison to both Uniform Cost Search and Iterative Deepening Search. A* Search focuses on expanding nodes that are believed to be on the most optimal route, instead of expanding all available nodes like UCS and IDS, this makes it a faster alternative to these searches. However, for complex iterations of the puzzle beyond start30, we begin to notice that A* Search is unable to sustain its memory cost. Therefore, while A* Search does optimize

memory better than UCS, it is still not efficient enough (in terms of memory) to tackle lengthy and complex searches.

The final Search method is Iterative Deepening A* Search and uses a combination of both Iterative Deepening Search and A* Search to resolve the shortcomings of A* Search, when it is faced up against complex and lengthy search tasks. Similar to regular Iterative Deepening Search, it only maintains the memory of its current search path. This makes it extremely memory efficient. Furthermore, using the heuristic from A* Search, it is able of expanding only the most promising and efficient nodes, which means it expands significantly less nodes than IDS. This allows IDA* to be extremely time efficient as well, thus allowing it to perform complex searches with both respectable memory and time efficiency.

Question 2

a)

	Start50		Start60		Start64	
IDA*	50	14642512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

b)

Code Changes

```
h(Node1, H1),
F1 is G1 + H1,
F1 =< F_limit,
depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

To

```
h(Node1, H1),
F1 is 0.8 * G1 + 1.2 * H1,
F1 =< F_limit,
depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

d)

All the 5 algorithms above show represent different variations of an objective function "*f(n) = (2-w)g(n) + wh(n)".* For IDA*, the weight (w) is 1 and for Greedy, the weight (w) is 2.

From the table of results obtained above, we can see that as we increase the weighting (w) and put more emphasis on h(n) for our results, the number of nodes expanded decreases significantly. We also notice that G, the length of the Path, begins to increase. Thus, we can see that we are sacrificing efficient pathing for algorithmic speed. This discrepancy becomes larger and larger as we move down the table until we reach Greedy Search, where we have completely eschewed the quality of our solution in favour of finding a possible solution as soon as possible. (For start50, the path with IDA* only has a length of 50, while the path for Greedy has a final length of 164).

Depending on our parameters, all these search algorithms have their own utility in real-world scenarios as some situations may favour accuracy, while others may emphasise speed or a mixture of both.

Question 3

a)

Another possible Heuristic that could be used in this scenario is the Manhattan Distance heuristic. This heuristic is the most optimal in a grid where there are only 4 directions of movement. The Manhattan Distance heuristic follows the given formula below.

$$h(x, y, x_G, y_G) = |x - x_G| + |y - y_G|$$

The Manhattan Distance heuristic calculates the absolute value of the horizontal and vertical distance to the goal and combines them. This provides a more accurate measure of the distance from the node to the goal than a straight line distance heuristic, as the costs for diagonal and horizontal/vertical movement are different and we are restricted to only horizontal and vertical movement.Thus, we are able to formulate a more suitable path to the goal.

b) i)

A heuristic is considered admissible when it never overestimates the cost of reaching a particular destination. Since a straight line is still theoretically the fastest way between two existing points, it is impossible to obtain a lower cost or lower number of moves than traversing

in a straight line. This means that the straight line distance heuristic does not overestimate the costs of reaching the goal node and it will still remain admissible in the case of diagonal movement.

ii)

The heuristic provided in part a, which was the manhattan distance, is no longer admissible. In some scenarios, the manhattan distance heuristic will overestimate the cost required to reach the goal and the cost that it estimates will be higher than the lowest possible cost achievable, now that diagonal movement is possible. For a heuristic to be admissible, this event must never occur.

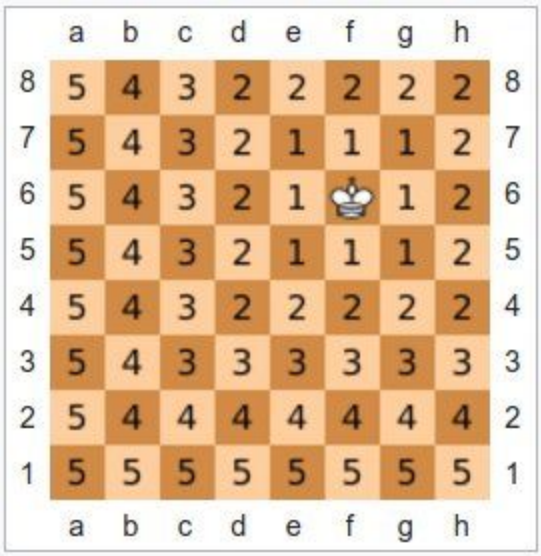
Let's assume a classical 3,4,5 triangle where 3 and 4 are the lengths of the adjacent and opposite side respectively, and 5 is the length of the hypotenuse. The current node and goal node are situated on opposite ends of the hypotenuse and there are arbitrary or no obstacles. Using the Manhattan distance formula, the cost would be 3 + 4 = 7, but using a straight line distance heuristic (which is possible, since movement can be diagonal), the cost would only be 5. Therefore, in this situation, the Manhattan distance heuristic has overestimated the minimum cost required to reach the goal node, thus making it inadmissible.

iii)

While the diagonal distance heuristic works excellently in scenarios where the agent is at a 45 degree angle from the goal, it becomes a bit more impractical in other more complex situations. Therefore the best possible admissible heuristic would be the chebyshev distance heuristic.

The chebyshev heuristic is used in chess and it calculates the distance between the agent and the goal as the maximum of the x-coordinate distance and y-coordinate distance as shown by the heuristic formula and image below.

$$h(x, y, x_{G,} y_G) = max\ (|x-x_G|, |y-y_G|)$$



As shown by the image above, with the numbers illustrating the costs to reach a particular square, to reach any particular point from a single position, the agent has to just move diagonally until it reaches either the row or column of the goal, and then move horizontally/vertically to the goal. This will thus be equivalent to the maximum of either the vertical or horizontal distance. This heuristic is admissible because it always returns the minimum number of steps required to reach a particular goal.

Question 4

a)

N	Combination	M	Maximum Speed (S)
1	+, -	2	1
2	+, 0, -	3	1
3	+, 0, 0, -	4	1
4	+, +, -, -	4	2
5	+, +, -, 0, -	5	2
6	+, +, 0, -, -	5	2
7	+, +, 0, -, 0, -	6	2
8	+, +, 0, 0, -, -	6	2
9	+, +, +, -, -, -	6	3

10	+, +, +, -, -, 0, -	7	3
11	+, +, +, -, 0, -, -	7	3
12	+, +, +, 0, -, -, -	7	3
13	+, +, +, 0, -, -, 0, -	8	3
14	+, +, +, 0, -, 0, -, -	8	3
15	+, +, +, 0, 0, -, -, -	8	3
16	+, +, +, +, -, -, -, -	8	4
17	+, +, +, +, -, -, -, 0, -	9	4
18	+, +, +, +, -, -, 0, -, -	9	4
19	+, +, +, +, -, 0, -, -, -	9	4
20	+, +, +, +, 0, -, -, -, -	9	4
21	+, +, +, +, 0, -, -, -, 0, -	10	4

b)

We are given the identity,

$$\lceil 2\sqrt{n} \rceil = \begin{cases} 2s+1, & \text{if } \quad s^2 \quad < n \leq s(s+1) \\ 2s+2, & \text{if } s(s+1) < n \leq \ (s+1)^2 \end{cases}$$

From the table presented above, we are able to discern that this case is true to some extent. For example, when N = 2, M = 3 and S = 1, since S(S+1) = 2 and N <= 2, then we use 2S + 1, which equals 3 (which is M). As we progress through the table, we notice that this formula applies to almost every value in the table except for a few as I will discuss below.

For specific values of N, where N = 1, 4, 9, 16, 25, 36 etc. the formula is unable to give us a correct solution. For example, when N = 4, M = 4 and S = 2. Substituting these numbers into the formula, we are able to draw that since S(S+1) = 6 and N = 4, then we should use the formula where M = 2S + 1. However, 2S + 1 = 5, whereas the actual optimal M value we can obtain is 4. Thus we can see that this formula does not work for square numbers.

Thus, this formula can be amended by adding a particular case for square numbers specifically, such that when maximum speed is the exact square root of N, then M must be 2N, in

accordance with the data obtained above. So the ideal formula would also include the statement,

M = 2S if S^2 = N

c)

If we assume that,

$$k \geq 0 \text{ and } n \geq \tfrac{1}{2}k(k-1)$$

Then we can deduce that reaching the goal with a velocity of 0 is possible. This is because ½(K(K-1)) will be the exact distance we require to decelerate from a velocity of K, assuming the added velocity is applied before the step and not afterwards.

Furthermore, for the base case where k = 0, we can clearly see that

$$M(n,0) = \left\lceil 2\sqrt{n} \right\rceil$$

Would return the same result as

$$M(n,k) = \left\lceil 2\sqrt{n + \tfrac{1}{2}k(k+1)} \right\rceil - k$$

Since N + ½(0(0+1) = N, so both results would be the same.

Now, if K > 0, then that would mean that the current velocity of the ‘vehicle’ is higher than 0. Since M is the total number of states (time changes), by starting off at velocity K, we do not need to expend K steps to accelerate to the velocity K. Therefore, we would thus require K less state changes than the base formula. Thus, we would have to add an additional ‘-K’ component to our original formula.

Now, since N is the distance required to reach our goal, we must also consider the distance travelled during the period where we have accelerated K steps. For example, the distance we need to travel to reach a velocity of 3 is 1 + 2 + 3 = 6. The distance we need to travel to reach a velocity of 4 is 1 + 2 + 3 + 4 = 10 and the distance we need to travel for a velocity of 5 is 1 + 2 + 3 + 4 + 5 = 15. The relationship between velocity and the distance travelled can thus be expressed as ½(K(K+1)). So we can add that to the N value in our original formula if we were to start off at a velocity of K. As a result, this would leave us with the final formula of:

$$M(n,k) = \left\lceil 2\sqrt{n + \tfrac{1}{2}k(k+1)} \right\rceil - k$$

d)

From what we have calculated above, we know that ½(K(K-1)) is the exact distance required to decelerate from a velocity of K (if the velocity is applied before the step) and reach our goal with a velocity of 0. Therefore if we assume the following parameters,

$$k \geq 0 \text{ and } n < \tfrac{1}{2}k(k-1)$$

Then it would be theoretically impossible to decelerate from a velocity of K and reach the goal with a velocity of 0. This is unless we slow to a stop, then turn around (or reverse) back to the goal position. In this case, firstly, we would have to calculate the number of steps it takes to reach a velocity of 0 from a velocity of K, and then, we have to calculate the number of steps it takes to backtrack from the destination we reached when our velocity became 0, back to the goal. We would then add both these portions together to obtain our final number of time steps required to reach the goal at a velocity of 0.

For the first portion, we can consider it logically. A vehicle travelling at a velocity of K, took K steps to reach that velocity and also K steps to decelerate from that velocity. Since we can disregard the K steps taken to reach the velocity K (since we are already at that velocity), then all we need to consider is that it takes K steps to decelerate from a velocity of K to 0. Thus the total number of steps required for the first portion is simply K.

Now for the second portion, the formula we need to consider is (albeit with some modifications):

$$M(n, 0) = \lceil 2\sqrt{n} \rceil$$

This is the formula used to find the number of steps required to reach a goal that is at distance of N from our current location, with our initial starting velocity being 0. This is applicable, because when we reach the end of our first portion, our velocity has become 0 and the N value would be how far we need to backtrack back to the goal. Thus we will be required to modify our N value in this formula.

So, first we need to find the optimal amount of distance required for a vehicle travelling at an initial velocity of K to slow to a stop. From earlier, we discovered that the distance required for a vehicle at a velocity of K to slow to a stop is ½(K(K-1). Now, to find out how far we need to backtrack back to the goal, we take that formula and subtract N from it. And thus, we can replace the N value in the formula with ½(K(K-1)) - N instead, as this will be how far we need to travel backwards to reach our goal.

In conclusion the final formula will be the conjunction of both portions discussed above.

$$M(n, k) = k + ceiling[2\sqrt{\tfrac{1}{2}k(k-1) - n}\,]$$

e)

The game itself is 2-dimensional, and the state of the agent's *(x,y)* position is *(r,c)* and their *(x,y)* velocity is *(u,v)*. Since the vehicle itself can move diagonally as well as horizontally/diagonally, we can use a modification of the chebyshev heuristic, which calculates the distance as the maximum of the steps in the x-coordinate and y-coordinate from the goal. The chebyshev heuristic is shown below,

$$h(x, y, x_G, y_G) = max\ (|x - x_G|, |y - y_G|)$$

Therefore we can split our equation into two components as provided by the hint, which stated that the heuristic would be of the form,

$$h(r, c, u, v, r_G, c_G) = \max(M(\ldots, \ldots), M(\ldots, \ldots))$$

To work out the number of steps required for the x-coordinate component, we must first find out the distance travelled on the x-axis (N). This would be $r_G - r$. We cannot use $|r - r_G|$ as shown in the chebyshev heuristic, because our velocity can be positive or negative, so the sign of N is important (if the vehicle is initially moving in the opposite direction to the goal). The initial x-axis velocity (K) would be u, as stated above.

For the y-coordinate component, we would have to find the distance travelled along the y-axis. This would work out to be $c_G - c$, and the initial velocity in the y-axis direction is v. Which brings us to our final heuristic of:

$$h(r, c, u, v, r_G, c_G) = \max(M(r_G - r, u), M(c_G - c, v))$$

This heuristic will always be admissible, since it will always be in the worst-case scenario (a straight horizontal/vertical line), the minimum number of steps required to reach the goal state, as reflected by the discussion of the chebyshev distance heuristic previously.