Tianqi Hu A99077299 ZhouHang Shao A99086018

Table 1:

h\o	0.0	0.1	0.2	0.3	0.4	0.5
0.0	<24995011, 9800>	<22493284, 9800>	<19955495, 9800>	<17197333, 9806>	<11143579,14490>	<-1,-1>
1.0	<6125049, 9800>	<6695680, 9800>	<3212647, 9800>	<1214569, 9806>	<10237660,14490>	<-1,-1>
2.0	<9800, 9800>	<10946, 10402>	<13561, 10840>	<19916, 11232>	<268630, 16246>	<-1,-1>

Abstract:

The key trade off between Dijkstra and A start algorithm is the speed and accuracy. Dijkstra can guarantee to detect the shortest path from one vertices to any other vertices in a non-negative graph. On the other hand, by using heuristic estimation for every calculation, A start algorithm can also find the shortest path given a source and a destination in a graph if the heuristic factor h = 1.0. Although A start algorithm can run faster than Dijkstra in general, as the function relies more on heuristic estimation, the returning path might be suboptimal. In other words, although heuristic estimation suppose to lead the algorithm to find shortest path, if it is relatively high then it would lead astray and return suboptimal solution.

Observation:

1. When o = 0.5, since the obstacle density is too high, no path could reach the destination.
2. When h = 0.0, the algorithm simply performs Dijkstra which can guarantee to find shortest path but in a slow speed
3. When h = 1.0, the algorithm can also find the shortest path and in a faster speed
4. When h = 2.0, the algorithm find the solution faster than h = 1.0 but lose more accuracy as number of obstacles in graph increases

Qualitative Analysis:

From the table above, as h = 0.0, the algorithm essentially run Dijkstra on the graph. Since Dijkstra is a greedy algorithm that will find the shortest path from source to any other coordinates, it will randomly “roam” the graph until the destination has been detected. Therefore, since the Dijkstra does not know exact location of the destination, it would explore more coordinations until the destination is found. Based on the table, the number of explored vertices for Dijkstra is 10 fold than h = 1.0 and 10^3 fold than h = 2.0. Therefore, Dijkstra will run relatively slow even though it can find the shortest path from the source to the destination.

However, by using Manhattan method as heuristic “guide” for finding the shortest path to destination, A start algorithm would explore less coordination. Since the algorithm will follow the

priority of f_score for the coordinate, h value will favor the coordinate which is closer to the destination. Thus, in general, A start is faster than Dijkstra.

Based on the observation table, when h = 1.0, which is the lower bound for A start algorithm, the shortest path can always be detected with less number of explored vertices. The reason for that is, the h factor simply provides the algorithm a “guide: to find the destination. Since moving from one coordinate to another cost 1 unit, which cost 1 unit closer to the destination, the heuristic calculation “compensate” that moving and thus cause the algorithm favor destination’s direction and at the same time guarantee that the algorithm will find the shortest path to the destination. In other words, since f_score = g_score + h_score, when h = 1.0, g_score and h_score equally contribute to f_score meaning that the algorithm will not only prefer the coordinate that is closer to the destination (heuristic estimation) but also consider the accumulated distance so far (Dijkstra part). Therefore, when h = 1.0, comparing to Dijkstra, the A start algorithm simply omit a large portion of explored vertices that don’t need to be explored as we can see from the table. Also, The balance between g and h let A start algorithm act as a perfect hybrid that could find the optimal solution in a faster speed.

However, if we increase h factor to 2.0, the heuristic calculation will play an more important role in finding the shortest path. In other words, since the algorithm enlarges the factor of the location of the destination, it becomes more “greedy” in terms of reaching the destination. Since larger h value provides a greater tendency toward the destination, the number of explored vertices decreases significantly. However, the tradeoff here is that the algorithm uses speed to sacrifice accuracy. From the extreme case, where h = 0.0 and o = 0.0, the A start algorithm find the destination without “wasting” any number of explored vertices (i.e number of explored vertices = shortest path) since h(n) provides a great tendency leading the algorithm to the destination. But as the obstacle density increases, since the algorithm “blindly” follow the direction, it will trying to reach the destination by detouring whenever it encounter obstacles. However, it would causes the loss of accuracy for finding the shortest path as we can see from the table that when 0.0 < o < 0.5 and h = 2.0, the algorithm will only find suboptimal solution. Therefore, the more “greedy” strategy causes the algorithm always return a suboptimal distance when the obstacle’s density is non zero. However, this strategy might be useful if we just want to find a relatively short path but in a certain amount of time.

Conclusion:

As we known that, Dijkstra is essentially a modified version of Breadth-First-Search dealing with graph that have various edge weight. Since Breadth-First-Search is not informed, Dijkstra inherit the same property as discussed before that Dijkstra could not know the location of the destination. However, by introducing heuristic calculation, Dijkstra can be modified as a informed algorithm or greedy-Best-First-Search just like A star algorithm. However, as we found previously, the greedy-Best-First-Search can only find the optimal solution when the cost of each moving equals to heuristic calculation. As the algorithm relies more on heuristic function, the returning solution would be suboptimal especially when the searching difficulties increases (i.e the number of obstacles)