COMP3411/9414/9814 Artificial Intelligence Session 1, 2018

Assignment 2 – Heuristics and Search

Student ID：z5142629

Student name: Wenxiao Xu

Date: 26th April 2018

Question 1: Search Algorithms for the 15-Puzzle

Answer (a):

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

Answer (b):

1) UCS:

According to the table above, the method of UCS is the worst in this case. The main Performance is running out of memory (except start10). This is because UCS always expands least-cost unexpanded node and it is known that all arc costs are 1 (equal). It has same efficiency with breadth-first search, which costs O(b^{d}) of time and spaces (expanded nodes). As a result, it spends too much memory to record all nodes (completely search).

2) IDS:

We know that the algorithm of IDS is built on the basis of DFS so that it only will cost linear spaces for expanded nodes (O(bd), better than UCS in this case), in other words, saving lot of memory. However, its time efficiency still is O(b^d) which causes the last two results of running out of time on the table.

3) A*:

A* search is combination of UCS and Greedy search, with respect to generated nodes, the time efficiency is exponential. We can see from the result table that A*'s (under admissible Heuristic h()) obviously is less than UCS's and IDS's. From the point of memory usage, although it only will expand optimal nodes, it still needs keep all nodes to store the scores of the heuristic function. With much more nodes needed to be stored, the memory usage becomes much bigger in order that running out of memory in last two search attempts(start30,40).

Moreover, compared with the third number of A* (915), it is less than the number of real minimum expanded nodes (952). Therefore, it may not be completed in this case.

4) IDA*:

IDA* is a low-memory variant of A* with pre-defined threshold. It is noticed that the expanded nodes results of IDA* are equal to the minimum number of moves recorded in the original code (also it is the only one get the result for all different start points in limited time with same memory condition), which means this solution is the least-cost one (expanded nodes for shortest path). It efficiently decreases the memory usage for A* with the appropriate cut-off point of heuristic function score (release spaces of that nodes exceeding the threshold).

Overview, IDA* is the most efficient search algorithm in this case.

Question 2: Heuristic Path Search for 15-Puzzle

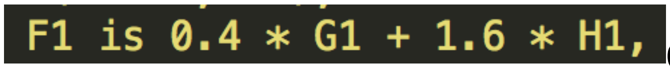
Answer (a/c):

	start50		start60		start60	
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

Answer (b):

According to the tutorial on week 4, f(n) = (2-w) * g(n) + w * h(n), where 0 <= w <= 2. As we known, when w = 0, f(n) = 2 g(n), which is a UCS; when w = 1, f(n) = g(n) + h(n), which is the standard A* search; when w = 2, f(n) = 2(h), which is a greedy search.

Therefore, if w = 1.2/1.4/1.6, the heuristic function would be , f(n) = 0.8/0.6/0.4 * g(n) + 1.2/1.4/1.6 * h(n). The changed code in 'heuristic.pl' would be shown below.



```
F1 is 0.4 * G1 + 1.6 * H1,
```

(w = 1.6)

Answer (d):

From the first row of IDA* to the fifth row of Greedy search, solutions of the length of the path increase but the total number of states expanded during the search decrease. The difference between five algorithms is the proportion of h()(selecting next node n which is closest to the goal). We can know the minimum numbers of moves from original codes are same with the result of using IDA* which are optimal answers (shortest path) of the length of the path. Therefore, except IDA*, other answers generated from other four algorithms are not the best solution. From the point of quality, the IDA is the best.

Moreover, although expanded nodes from the table are less than the result of IDA*, which looks like using less memory, the truth is all five algorithms need to store all nodes to keep the record scores calculated from different evaluation function f(n) as the selection criteria.

From the point of speed, assuming each move cost equally, according to the increase of the number of nodes expanded, running time increase. Therefore, under the situation of sacrificing of finding the shortest solution(quality), as the increase of w(a.k.a proportion of h()) increases(from IDA* search to greedy search), the algorithm speeds up.

Question 3: Maze Search Heuristics

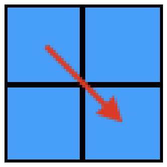
Answer (a):

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

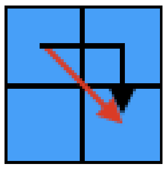
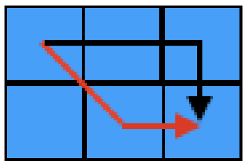
According to the constraint of declaration of this maze, in fact, it should be noticed that the move cannot cross nodes along the straight line (each move should like 8-puzzle, one step be one step so that the above formula is admissible). Therefore, considering the sum of horizontal and vertical moves as the heuristic h() will be always larger than the Straight-Line-Distance heuristic, which means my formula dominates the previous straight line one(but will not overestimate the true cost because of the exist of obstacle of maze wall).

Answer (b):

i. The answer is no, because the straight-line distance will get √2 (actual move diagonally) which is larger than true cost of 1, which means this heuristic will overestimate the cost. Therefore, it is not admissible any more.



ii. The answer is no, my formula is in fact to calculate the Manhattan Distance. Because it cannot guarantee the heuristic cost will be always less than the true cost, actual move can be diagonal as 1, but Manhattan Distance is 2(as the picture shown below).

the red arrow is the true path, (if there is no obstacle at a distance of maze path) which will be less than the Manhattan distance. Therefore, it is not admissible any more.

iii. The answer is $h(x, y, x_G, y_G) = \max(|x - x_G|, |y - y_G|)$

Question 4: Graph Paper Grand Prix

Answer (a):

n	optimal sequence	+	-	o	M(n, 0)
1	+ -	1	1	0	2
2	+ o -	1	1	1	3
3	+ o o -	1	1	2	4
4	+ + - -	2	2	0	4
5	+ + - o -	2	2	1	5
6	+ + o - -	2	2	1	5
7	+ + o - o -	2	2	2	6
8	+ + o o - -	2	2	2	6
9	+ + + - - -	3	3	0	6
10	+ + + - - o -	3	3	1	7
11	+ + + - o - -	3	3	1	7
12	+ + + o - - -	3	3	1	7
13	+ + + o - - o -	3	3	2	8
14	+ + + o - o - -	3	3	2	8
15	+ + + o o - - -	3	3	2	8
16	+ + + + - - - -	4	4	0	8
17	+ + + + - - - o -	4	4	1	9
18	+ + + + - - o - -	4	4	1	9
19	+ + + + - o - - -	4	4	1	9
20	+ + + + o - - - -	4	4	1	9
21	+ + + + o - - - o -	4	4	2	10

Answer (b):

s(+/-)	s^2	s*(s+1)	(s+1)^2	2s+1	2s+2	M(n, 0)	ceil(2√n)	2√n	n
1	1	2	4	3	4	2	2	2	1
1	1	2	4	3	4	3	3	2.82	2
1	1	2	4	3	4	4	4	3.46	3
2	4	6	9	5	6	4	4	4	4
2	4	6	9	5	6	5	5	4.72	5
2	4	6	9	5	6	5	5	4.90	6
2	4	6	9	5	6	6	6	5.29	7
2	4	6	9	5	6	6	6	5.66	8
3	9	12	16	7	8	6	6	6	9
3	9	12	16	7	8	7	7	6.32	10
3	9	12	16	7	8	7	7	6.63	11
3	9	12	16	7	8	7	7	6.93	12
3	9	12	16	7	8	8	8	7.21	13
3	9	12	16	7	8	8	8	7.48	14
3	9	12	16	7	8	8	8	7.75	15
4	16	20	25	9	10	8	8	8	16
4	16	20	25	9	10	9	9	8.25	17
4	16	20	25	9	10	9	9	8.49	18
4	16	20	25	9	10	9	9	8.72	19
4	16	20	25	9	10	9	9	8.94	20
4	16	20	25	9	10	10	10	9.17	21

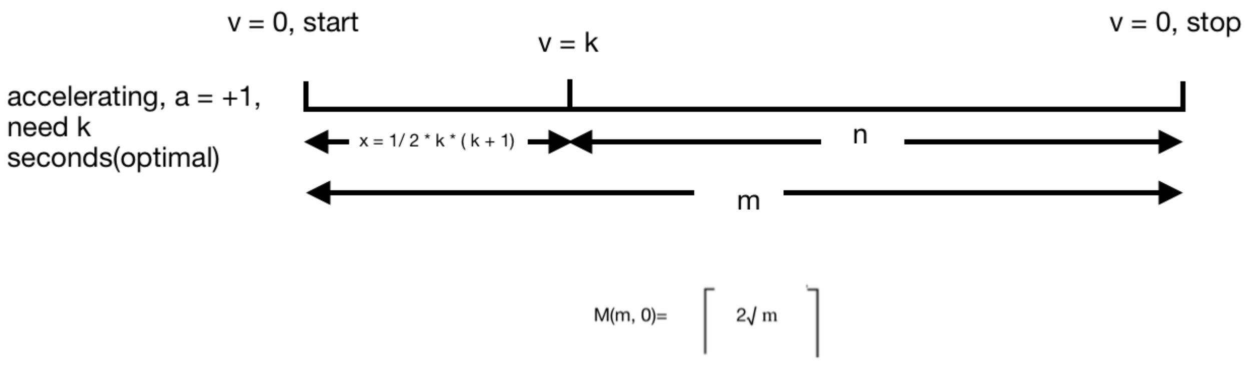
It is noticed that in order to getting the goal position at velocity of 0 (k = 0), the operations of increase and decrease must be token in pairs. Moreover, the number of 'o' in optimal sequence is in value range of {0, 1, 2}. Assuming we assign the number of '+'(or '-') as s, according to the value of n, we could find some special points (n is a square number) where the rows of table fill up with grey color(s = √ n, #o is 0, M(n,0) = 2s). From the above table, we can see that $s(s+1)$ as the boundary divides equally the amount of n into 2 parts (where $s^2 < n <= s(s+1)$ and $s(s+1) < n <= (s+1)^2$), the optimal sequence of the former part just has 1 'o', the latter has 2 'o's. As a result, M(n,0) would be equal to 2s + 1 or 2s + 2 (2s means 1 pair of '+/-') where the table shows with red number belonging to the range of yellow space, which is corresponding to this identity:

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

(because the formula on LHS needs to be rounded up, which is decided by decimal part of √ n. if decimal part >= .5, multiplied by 2 results in + 2 in total, otherwise + 1)

Answer (c):

if k >= 0 and n >= (1/2 * k * (k−1)) then:



From the picture we assume that, firstly agent needs k second to accelerate to the velocity of k as a result of getting the point of the position (x, k). Therefore, the distance of x is ½ * k * (k+1). Also, from the beginning of the start to the stop position, the total distance is m so that m = x + n, where n is the distance of starting with velocity k given by the title. At the end, as we known that the total m distance needs M(m,0) second, which equals to the ceiling of 2 √ m (from question (b)). Therefore, substituting n and x into the formula of M(m,0), we could get:

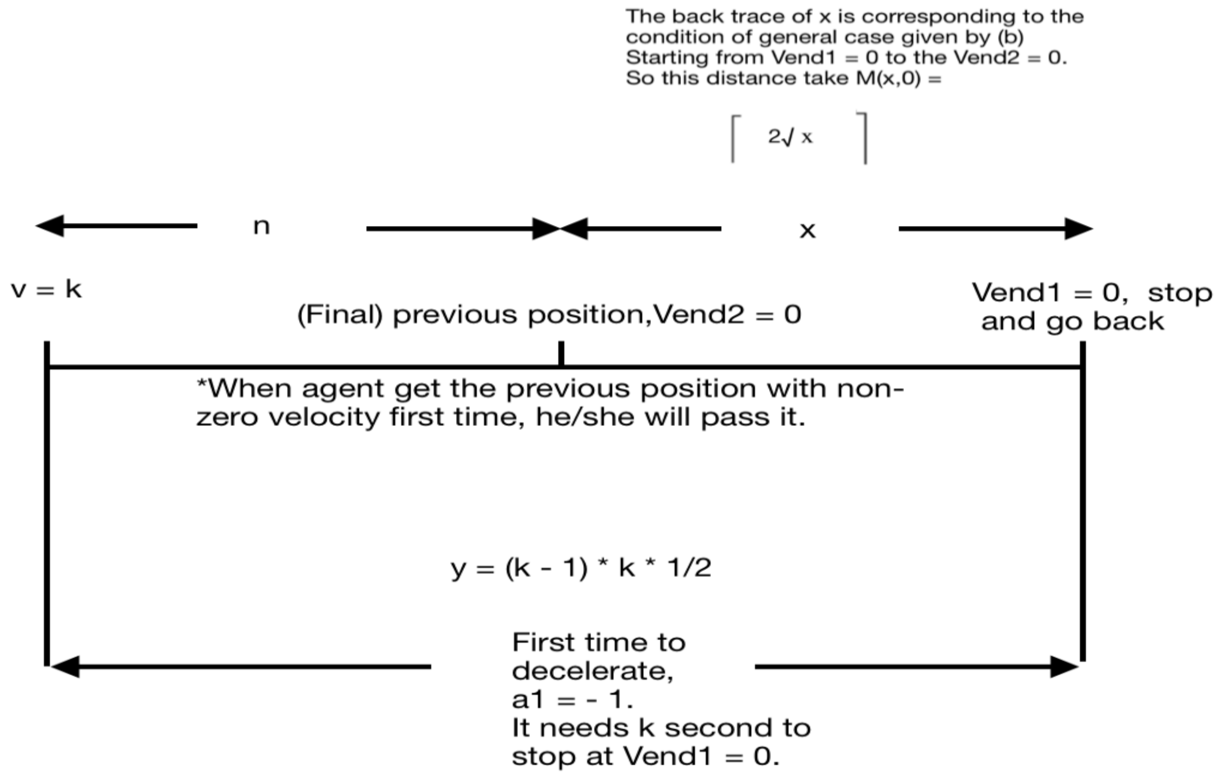
M(m, 0)= $\left\lceil 2\sqrt{n + \frac{1}{2}k(k+1)} \right\rceil$

we know it needs k second to accelerate at the beginning, therefore, after minus k, we get:

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

Answer (d):

If n < ½ * k * (k – 1), the agent will overshoot the previous stop position (can not stop immediately). The situation shows as below:


The back trace of x is corresponding to the condition of general case given by (b) Starting from Vend1 = 0 to the Vend2 = 0. So this distance take M(x,0) =
⌈ 2√ x ⌉
n
x
v = k
(Final) previous position,Vend2 = 0
Vend1 = 0, stop and go back
*When agent get the previous position with non-zero velocity first time, he/she will pass it.
y = (k - 1) * k * 1/2
First time to decelerate, a1 = - 1. It needs k second to stop at Vend1 = 0.

If starting with v = k, it needs first k second to stop at Vend = 0. Then, it also needs the time of the ceiling of 2 √ x to go back to the previous final position. As shown on picture, when agent first time decelerate until stop (overshoot the final position), the accelerate equals to -1 so that it costs y = ½ * (k – 1) * k distance. We have known that the final position from the beginning with velocity of k is n, which derives that x = y – n. As a result, the agent will experience two time period (k + M(x,0)), substituting x and y into formula, finally we get:

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

Answer (e):

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