COMP3411/9414/9814 Artificial Intelligence Session 1, 2018

Assignment 2 – Heuristics and Search

Name: Xiaodan Wang

Student ID: z5145114

Question 1: Search Algorithms for the 15-Puzzle

(a)

Number of Nodes	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 above, it is clearly to see the time and space efficiency of four different searches.

Compared with Iterative Deepening Search, Uniform Cost Search takes more memory usage as it may face memory problem when deal with complex issues. The space property of Uniform Cost Search is $O(b^d)$, while Iterative Deepening Search's is $O(bd)$. So with the increasing of depth of the least-cost solution, memory usage of Uniform Cost Search raises dramatically. In addition, Uniform Cost Search and Iterative Deepening Search has the same time efficiency which is $O(b^d)$.

A* Search preserves efficiency of Greedy Search and use heuristic to avoid expanding paths that are already expensive. In this way, A* Search is more efficient than two above. The time and memory usage is less as it does not expand every path. However, it still is facing the space problem dealing with complex issues because A* Search keeps all nodes in memory.

Iterative Deepening A* Search is a low-memory variant of A* Search by combining Depth First Search. So it takes less time and space than above three.

In conclusion, Iterative Deepening Search and Uniform Cost Search has same time efficiency and Uniform Cost Search has more memory usage. A* Search is more time-efficient and space-efficient than above two but still face the time problem. Iterative Deepening A* Search is the most time-efficient and space-efficient method among these four searches.

Question 2: Heuristic Path Search for 15-Puzzle

(a)

	start50		start60		start64	
Greedy	164	5447	166	1617	184	2174

(b)

The code to be changed.

```
31  % Otherwise, use Prolog backtracking to explore all successors
32  % of the current node, in the order returned by s
33  % Keep searching until goal is found, or F_limit is exceeded.
34  depthlim(Path, Node, G, F_limit, Sol, G2)  :-
35      nb_getval(counter, N),
36      N1 is N + 1,
37      nb_setval(counter, N1),
38      % write(Node),nl,   % print nodes as they are expanded
39      s(Node, Node1, C),
40      not(member(Node1, Path)),      % Prevent a cycle
41      G1 is G + C,
42      h(Node1, H1),
43      F1 is G1 + H1,
44      F1 =< F_limit,
45      depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

In week 4 Tutorial Exercise, the heuristic path algorithm's objective function is $f(n)=(2-w)g(n)+wh(n)$.

Change the code in line 43 "F1 is G1 + H1".

When w=1.2, $f(n)=0.8*g(n)+1.2*h(n)$. Change it into "F1 is G1*0.8 + H1*1.2".

```
depthlim(Path, Node, G, F_limit, Sol, G2)  :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node),nl,   % print nodes as they are expanded
    s(Node, Node1, C),
    not(member(Node1, Path)),      % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is G1*0.8 + H1*1.2,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

When w=1.4, $f(n)=0.6*g(n)+1.4*h(n)$. Change it into "F1 is G1*0.6 + H1*1.4".

```
depthlim(Path, Node, G, F_limit, Sol, G2)  :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node),nl,   % print nodes as they are expanded
    s(Node, Node1, C),
    not(member(Node1, Path)),      % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is G1*0.6 + H1*1.4,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

When w=1.6, $f(n)=0.4*g(n)+1.6*h(n)$. Change it into "F1 is G1*0.4 + H1*1.6".

```
depthlim(Path, Node, G, F_limit, Sol, G2)  :-
    nb_getval(counter, N),
    N1 is N + 1,
    nb_setval(counter, N1),
    % write(Node),nl,   % print nodes as they are expanded
    s(Node, Node1, C),
    not(member(Node1, Path)),      % Prevent a cycle
    G1 is G + C,
    h(Node1, H1),
    F1 is G1*0.4 + H1*1.6,
    F1 =< F_limit,
    depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

(c)

	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

(d)

Among five algorithms, IDA* has the best quality of solution as it always finds the shortest solution. With the increasing of *w*, the quality of solution decrease, as well as Greedy is the last.

However, considering speed, Greedy expends least nodes so it is more time-efficiency than others. IDA* takes longest time. With the changes of w, time-efficiency is uncertain as heuristics function works sometime.

Question 3: Maze Search Heuristics

(a) $h(x, y, x_G, y_G) = |x\text{-}x_G| + |y\text{-}y_G|$

(b)

(i) Yes, the Straight-Line-Distance heuristic is still admissible.

As $h_{SLD}(x, y, x_G, y_G) = \sqrt{(x - x_G)^2 + (y - y_G)^2}$ is the shortest distance between two position, it costs least moves from current position to goal position.

Although the agent can take step diagonally, the true cost will not smaller than the Straight-Line-Distance heuristic.

(ii) My heuristic $h(x, y, x_G, y_G) = |x\text{-}x_G| + |y\text{-}y_G|$ is not admissible anymore.

The reason is that my heuristic doesn't considerate the diagonally move case. In my heuristic, if the agent wants to reach the diagonal position, it has to move left or right first, then move up or down, which costs 2 moves. However, the agent can move diagonally so the true cost is 1 move.

In this case, true cost will less than my heuristic when the agent could move diagonally. So my heuristic is not admissible anymore.

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

Question 4: Graph Paper Grand Prix

(a)

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

(b)

(i) First of all, start with the special case $n = s^2$.

The agent could move with increasing velocity on first half and move with decreasing velocity on last half.

To prove that:

For the velocity keeping-increase period, the agent moves from location 0 to R, where R is

$$R = 1 + 2 + \cdots + s = \frac{(1+s)\times s}{2} = \frac{s^2 + s}{2}$$

So the agent reaches the location $R = \frac{s^2+s}{2}$ with velocity is s. And it takes s moves.

For the velocity decreasing period, the agent moves from location R to the end location R_{end}, where

$$R_{end} = R + (s-1) + (s-2) + \cdots + 1 + 0 = \frac{s^2 + s}{2} + \frac{(s-1+0)\times s}{2} = \frac{2s^2 + s - s}{2} = s^2$$

Obviously, end location R_{end} is Goal location n. And it takes s moves.

$$s^2 = n$$

The number of total moves is

$$s + s = 2s, \quad \text{for } n = s^2$$

(ii) As $s^2 < n < (s+1)^2$, it will take some rests "o" in the decreasing period to reach the goal location.

From (i), the agent's highest velocity is s. So if it rests once, the furthest distance the agent could reach is

$$R_{max} = 2s + s$$

when the velocity keeps on *s* once and then decreases.

In this case,

$$R_{max} = s(s+1) \geq n$$

So when $s^2 < n \leq s(s+1)$, the number of moves is

$$2s + 1$$

which is total moves from (i) plus one more rest move.

In another case, when $s(s+1) < n < (s+1)^2$, it takes two rests "o".

So the number of moves is

$$2s + 2$$

In conclusion, when $n = s^2$,

$$M(n, 0) = 2s$$

when $s^2 < n \leq s(s+1)$,

$$M(n, 0) = 2s + 1$$

when $s(s+1) < n < (s+1)^2$,

$$M(n, 0) = 2s + 2$$

As

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

the $M(n, 0)$ stratifies

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

(c)

From (b), we prove that the number of moves from location 0 with velocity 0 to location *n* with velocity 0 is $M(n, 0) = \lceil 2\sqrt{n} \rceil$.

So it is easy to get the number of moves from location 0 with velocity *k* to location *n* with velocity 0 when considering it is a part of larger path.

If the agent goes right directly, to reach the velocity 0, the agent has to move a path which length is R_{min} at least. While the agent decreases velocity on every movement,

$$R_{min} = (k-1) + (k-2) + \cdots + 0 = \frac{k(k-1)}{k}$$

Consider a larger path which the agent moves from location *-R1 (R1>0)* with velocity 0 to location *n* with velocity 0. Obviously, it will pass the location 0 and we can assume its velocity is *k* at location 0. It is easy to get

$$M(n,k) = M(R1+n,0) - M_{extra}$$

where M_{extra} is the number of movements from location *-R1* with velocity 0 to location 0 with velocity k.

As well as, the shortest path is increase the velocity on every movement, so

$$M_{extra} = k$$

To calculate *R1*, the agent moves from location *-R1* with velocity 0 to location 0 with velocity k. As the agent increases its velocity on every movement,

$$R1 = 1 + 2 + \cdots + k = \frac{k(k+1)}{2}$$

Finally, get

$$M(n,k) = \lceil 2\sqrt{R1+n} \rceil - k = \left\lceil 2\sqrt{n + \frac{k(k+1)}{2}} \right\rceil - k$$

(d)

In another case, when $k \geq 0$ and $n < \frac{k(k-1)}{2}$, the agent will overshoot the goal position if it keeps going right. So the agent has to reverse the goal position. In this path, the agent moves from location 0 with velocity k to location *R2 (R2 > n)* with velocity 0 first, then moves from location *R2* with velocity 0 to location n with velocity 0. It can be calculated as

$$M(n,k) = M(R2,k) + M(R2-n,0)$$

To calculate R2, the shortest one is keeping decrease in every movement, so the length of path is

$$R2 = (k-1) + (k-2) + \cdots + 1 = \frac{k(k-1)}{2}$$

Finally, get

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

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

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