COMP9414 Assignment 2 Heuristics and Search

- **Yunqiu Xu**
- **z5096489**
- **yunqiuxu1991@gmail.com**

Question 1 Maze Search Heuristics

(a)

- Another admissible heuristic will be total "Manhattan Distance" which calculate the distance in each direction:

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

- Heuristic based on Manhattan Distance is admissible:
  - If if there is no obstacle, the cost of path will be equal to Manhattan Distance from start position to the goal
  - If there are obstacles, the cost will also be no less than the heuristic
  - Thus the cost of path will not be shorter than Manhattan Distance $\Rightarrow$ Admissible
- Manhattan Distance heuristic dominates Straight-Line-Distance heuristic:
  - The shortest distance between two points is a straight line
  - For every node n, the cost of Manhattan Distance heuristic will not be shorter than the cost of Straight-Line-Distance heuristic. Because the sum of two edges of a triangle is larger than another edge.
  - Thus Manhattan Distance heuristic dominates Straight-Line-Distance heuristic.

(b)

(i)

- The Straight-Line-Distance heuristic is not admissible.

- Suppose we will move from $(0,0)$ to $(3,4)$
- We can move from $(0,0)$ to $(3,3)$ first, and the cost is 3 moves
- Then we move from $(3,3)$ to $(3,4)$, and the cost is 1 move
- Thus the total cost is 4 moves, however by using Straight-Line-Distance heuristic we need to spend at least 5 moves
- So in this conditions the cost of some paths will be less than Straight-Line-Distance heuristic $\Rightarrow$ not admissible

(ii)

- The Manhattan Distance heuristic is not admissible:
  - We will use the same example in (i) : move from $(0,0)$ to $(3,4)$
  - From (i) we can know that the shortest path only cost us 4 moves
  - However by using Manhattan Distance heuristic the cost is 7 moves
  - Thus Manhattan Distance heuristic is not admissible for that it overestimates the cost to reach the goal

(iii)

- The best heuristic can be Chebyshev Distance heuristic:

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

- This heuristic is admissible, to reach the goal we may cost at least the maximum of distance in one of the directions.

Question 2 Search Algorithms for the 15-Puzzle

(a)

Algorithm \ Start	**start10**	**start12**	**start20**	**start30**	**start40**
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Mem	Mem
A*	33	26	915	Mem	Mem
IDA*(Man)	29	21	952	17297	112571

Algorithm \ Start	start10	start12	start20	start30	start40
IDA*(Mis)	35	87	4345	2105465	Time

(b) puzzle15mis.pl using Count Misplaced Tiles heuristic

```
% modified totdist using Count Misplaced Tiles heuristic
% totdist([1/2,1/3,2/1,2/2,3/1,3/2],[1/2,1/3,2/2,2/1,3/1,3/2],D). --> D = 2
totdist([], [], 0).
totdist([Tile|Tiles], [Position|Positions], D) :-
    misplaced(Tile, Position, D1), %this can be used to compute misplaced tiles
    totdist(Tiles, Positions, D2),
    D is D1 + D2.

% function misplaced: compute the number of misplaced tiles
% misplaced(1/2, 1/2, D). --> D = 0
% misplaced(1/2, 1/3, D). --> D = 1
misplaced(X/Y, X1/Y1, D) :-
    X is X1,
    Y is Y1,
    D = 0, !;
    D = 1.
```

(c) Briefly discuss the efficiency of these five algorithms.

- Set b as the maximum branching factor of search tree
- UCS has the largest space complexity ($O(b^{\lceil C^*/\epsilon \rceil})$, C^* is the cost of optimal solution). Thus it's not effiecient and easy to go out of memory
- Compared with UCS, IDS takes less space complexity($O(bd)$, where d is shallowest depth to find the solution). It does not exceed memory at "start20". However there are still a large number of nodes ,the search will terminate in "start30" and "start40" due to space exceed.
- A* will concentrates on exploring the most promising nodes and will not traverse all the tree. It costs much less space than UCS and IDS in "start10", "start12" and "start20", however, for "start30" and "start40" it still needs too much memory.
- IDA* combines the advantages of A* and IDS, that it is asymptotically as efficient

as A* but uses lower memory. For IDA* using Manhattan Distance heuristic, the memory usage is similar or lower than A* that we can get the result of "start40" and "start50".

- For A* we need to choose suitable heuristic function. Count Misplaced Tiles is also an admissible heuristic. As each tile can only move to adjacent positions, Manhattan Distance gives the shortest distance, so it's dominated by Count Misplaced Tiles. When we use Count Misplaced Tiles heuristic, it costs more memory and longer time to compute. In "start40" it exceeds the time limit.
- In summary, A* Algorithm is more efficient than UCS & IDS, and IDA* costs lower memory. Different heuristic functions will lead to different performance, IDA* using Manhattan Distance heuristic performs better(less memory, shorter time) than the one using Count Misplaced Tiles heuristic.

Question 3 Heuristic Path Search for the 15-Puzzle

Algorithm / Start	**start50**	**start50**	**start60**	**start60**	**start64**	**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
1.8	240	35557	314	8814	344	2209
Greedy	164	5447	166	1617	184	2174

(a) Run [greedy] for start50, start60 and start64

(b) w = 1.2

```
% Modify IDA* to set w = 1.2
```

```
% f(n) = (2-w)g(n) + wh(n)

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),

    % I make modification here
    F1 is G1 + H1,    % This is the original version
    % F1 is 0.8 * G1 + 1.2 * H1,    w = 1.2
    % F1 is 0.6 * G1 + 1.4 * H1,    w = 1.4
    % F1 is 0.4 * G1 + 1.6 * H1,    w = 1.6
    % F1 is 0.2 * G1 + 1.8 * H1,    w = 1.8

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

(c) w = 1.4, 1.6, 1.8

(d) Briefly discuss the tradeoff between speed and quality of solution for these six algorithms.

- The objective function for heuristic path algorithm is like

$$f(n) = (2 - w)g(n) + wh(n)$$

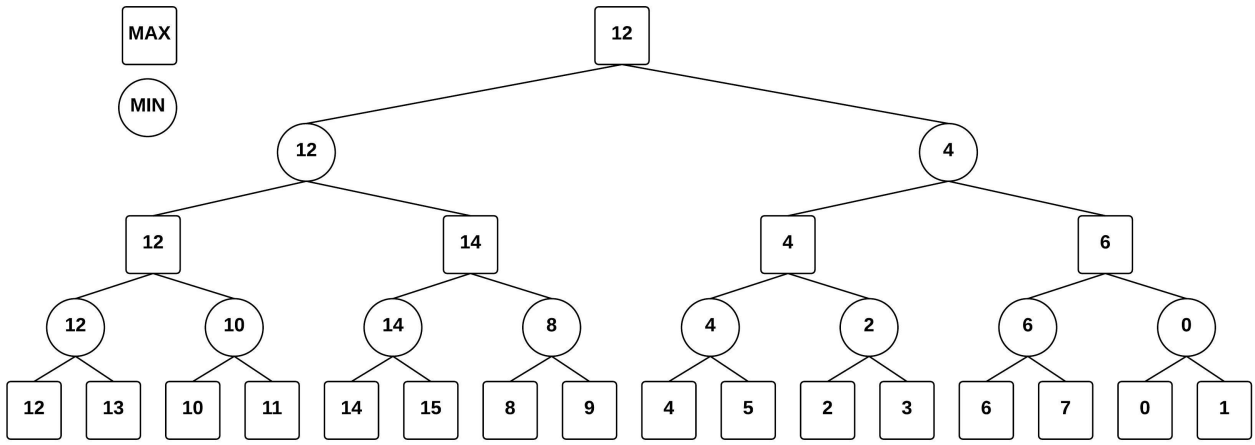
- In our original version we give same weight to g(n) and h(n), we can see that the all paths are short, while the number of nodes is so large that it costs us long time to finish the search.
- As we set larger w, the weight of h(n) increases, thus the algorithm is prone to greedy search. The time cost is decreasing but our algorithm is not as optimal as before for that the length of path increases. As the greedy search is not complete, our search may be stuck in loops that we will not expand new nodes. For example, in "start60" when w = 1.4, we can see that it costs only 4432 nodes which is much less than IDA* with other w values.

- When w is 1.8, where the weight of heuristic function is largest among all IDA* with different weights, we can see that our search is not optimal that the length of path is the largest of all other IDA* search in each column. Maybe the search is not complete in "start64" that the number of nodes is very small.
- Comepared with IDA*, the greedy method takes shortest time. However the search is not optimal that the length of path is large. The length of path is smaller than IDA*(w = 1.8), maybe this is because when w = 1.8($f(n) = 0.2g(n) + 1.8h(n)$) the number of nodes for h(n) is expanded larger than greedy method. Greedy method is incomplete that the number of nodes is smaller than any IDA* search in each column, which means our search may go into loops.

Question 4 Game Trees and Pruning

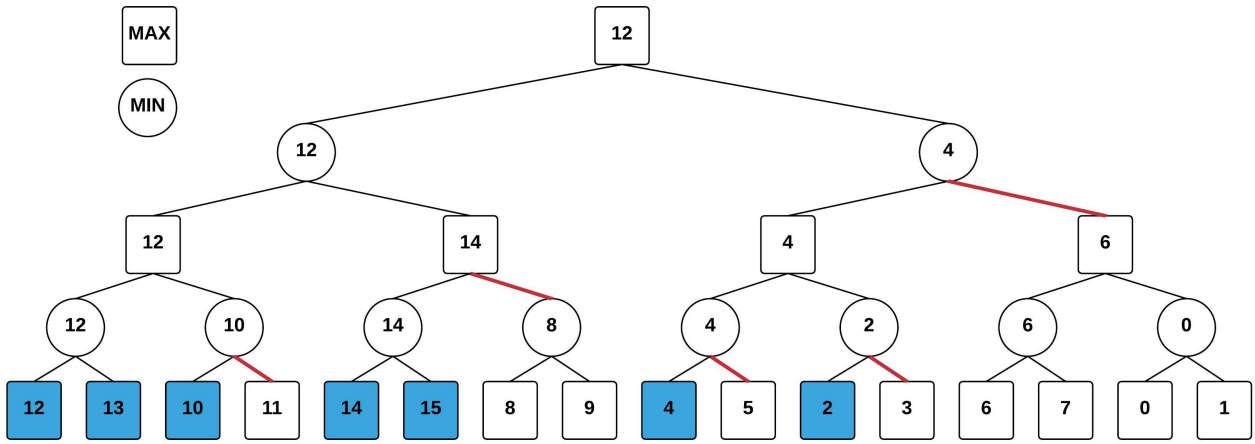
(a) Fill in the leaves of this game tree with all of the values from 0 to 15, in such a way that the alpha-beta algorithm prunes as many nodes as possible

- As the hint said, the left subtree is preferable, so we can make the tree in following way:
  - If current node is "max", its left child should be larger than right child thus the left child will be chosen
  - If current node is "min", its left child should be smaller than right child thus the left child will be chosen
- So the tree can be shown as:



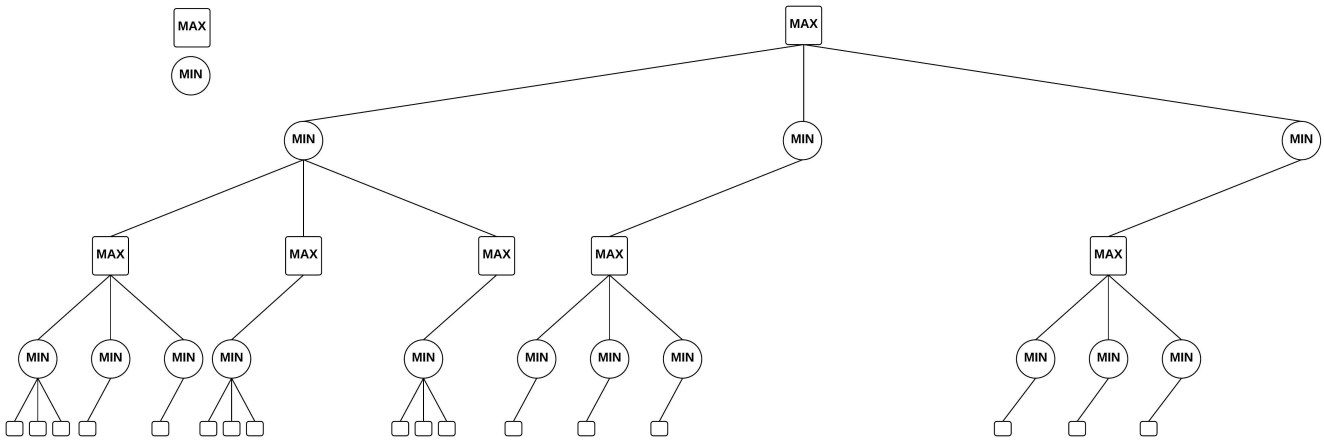
(b) Trace through the alpha-beta search algorithm on your tree. How many of the original 16 leaves are evaluated?

- 7 leaves are evaluated and filled as blue: 12, 13, 10, 14, 15, 4, 2
- Pruned branches are marked by red



(c) Each internal node has exactly three children, draw the shape of the pruned tree. How many of the original 81 leaves will be evaluated?

- 17 leaves will be evaluated



(d) What is the time complexity of alpha-beta search, if the best move is always examined first (at every branch of the tree)?

- Set b as branching factor, and d as the depth of tree
- If the best move is always examined first, the time complexity of alpha-beta search is $O(b^{\frac{d}{2}})$
- Explanation:
  - The time complexity of min-max search is $O(b^d)$
  - By using alpha-beta prune, for odd depth(find max) the best moves are always searched first, for even path(find min) the worst moves are always searched first

    +So the number of leaf node positions evaluated in odd depth is about $O(b * 1 * b * 1 * \ldots * b) \Rightarrow O(b^{\frac{d}{2}})$
  - And the number of leaf node positions evaluated in even depth is also $O(b * 1 * b * 1 * \ldots * 1) \Rightarrow O(b^{\frac{d}{2}})$
  - Thus we can take only half time of min-max search $\Rightarrow O(b^{\frac{d}{2}})$