COMP3411/9414 Artificial Intelligence Term 1, 2019
Assignment 2 – Heuristics and Search

Senlin Deng, z5014567

Question 1: Search Algorithms for the 15-Puzzle.

a). The table us showing as follow:

	start10	*start20*	*start27*	*start35*	*start43*
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	5297410	Time	Time	Time
A^*	33	915	1873	Mem	Mem
IDA^*	29	952	2237	215612	2884650

b).

UCS with Dijkstra: The main idea of this algorithm is similar to **Breadth-First Search**. It extends nodes at the head of queue by generating successors of its destination node. By using Dijkstra's algorithm, this process is more efficient. However, the expanded nodes can still be gigantically large to get the optimal solution. As we can tell from the above table, UCS is not able to solve the puzzle with a path length over that 20, due to running out of memory. The time complexity of this algorithm is $O(b^{\lceil C^*/\varepsilon \rceil})$, where C^* denotes cost of optimal solution and assume every action cost at least ε. As a result, the **space** of this algorithm could be **gigantically large** by solving sophisticated problems. And the **time complexity** is not **fairly efficient**, since it expands all "legs" for the current nodes.

IDS: This algorithm is an optimized version of **Depth-First Search**. This algorithm saves lots of memory comparing to UCS. It only stores the "current path". It uses depth first search to look for a solution recursively, up to the specified depth limit. As a result, this algorithm can **save the space effectively**. However, this algorithm is time-consuming. For each given depth limit, it recursively processes the nodes and the **time increases exponentially**.

A^*: This algorithm is an **informed search algorithm**. Compared to the above two algorithms, this algorithm records the **estimated costs** from **current** state to **goal**. Thus, this process skips many states which has a relatively larger estimated-cost. For example, when solving the problem with path a length of twenty, the number of expanded nodes is gigantically larger for UCS and IDS, while A^* only expands less than one thousand nodes. Thus**, this algorithm is much more efficient than uninformed search algorithm** (e.g. UCS and IDS) in space and time. However, it extends "leg" at head of queue by generating successors of its destination nodes and keeps all nodes in memory. As a result, the **space** of this algorithm could be **gigantically large** by solving sophisticated problems. And the time can **increase exponentially** (in relative error in heuristic times length of solution). The above table shows the result.

IDA*: From the above table, it can be concluded that this algorithm is the **most efficient** one compared to the rest of algorithms. This algorithm takes the benefits from IDS and A*. **IDA* is a low-memory variant of A*** which performs a series of depth-first search. It cuts off certain search when the sum of *f(n)* exceeds some pre-defined threshold. And the threshold is increased steadily until the problem is solved.

Question 2: Deceptive Starting States.

a). The heuristic values for states start49 and start51 are **25 and 43** respectively.

b). The number of expanded nodes is **551168**.

c). The reason why start49 expanded much more nodes than start51 is that **start51 has a relatively larger heuristic value**. The IDA* algorithm used the idea of deep limited search. It performs a series of depth-limited searches with increasing *F_limit*. The nodes are restricted with a $f(n) \leq F\_limit$. And this processing will not stop until **either goal is found or *F_limit* is exceeded**. Since the path length of these two states are similar, the *F_limit* will be pretty much the same for each iteration (e.g. *F_limit1 = F_limit + 2*). Since start51 has a relatively larger heuristic value and we have $\boldsymbol{f(n) = g(n) + h(n)}$**.** With the same value of *F_limit*, start49 need to expand much more nodes to get higher $g(n)$ to exceed certain *F_limit*. That is why they have similar path length but various expanded nodes.

Question 3: Heuristic Path Search.

a).

	start49		**start60**		**start64**	
IDA*	49	178880187	60	321252368	64	1209086782
Greedy	133	5237	166	1617	184	2174

b). The heuristic path algorithm is a best-first search in which the objective function is $\boldsymbol{f(n) = (2 - \omega) * g(n) + \omega * h(n)}$. With $\omega = 1.2$, We have $f(n) = 0.8 * g(n) + 1.2 * h(n)$. And we only need to change to the code in the **third last line** about how to compute *f(n)*. We change it from `F1 is G1 + H1,` to `F1 is 0.8*G1 + 1.2*H1,`, since our $\omega = 1.2$.

c).

	start49		**start60**		**start64**	
IDA*	49	178880187	60	321252368	64	1209086782
1.2	51	988332	62	230816	66	431033
1.4	57	311704	82	4432	94	190278
Greedy	133	5237	166	1617	184	2174

d).

Those four algorithms can be represented by the form that $\boldsymbol{f(n) = (2 - \omega) * g(n) + \omega * h(n)}$, ($\omega = 1$ for IDA* and $\omega = 2$ for Greedy). From the tutorial, this algorithm will generate optimal solution when $0 \leq \omega \leq 1$. Because this is equivalent to A star search using the heuristic $h'(n) = \left[\frac{\omega}{2-\omega}\right] h(n) \leq h(n)$. And it can be concluded from the above table that the **speed is getting faster while $\boldsymbol{\omega}$ is increasing**, because the number of expanded nodes is decreasing. But the **quality of solution is inversely proportional to $\boldsymbol{\omega}$**, since the length of path is increasing while ω is increasing. As a result, **fast speed produces low quality of solution**.

Question 4: Maze Search Heuristics.

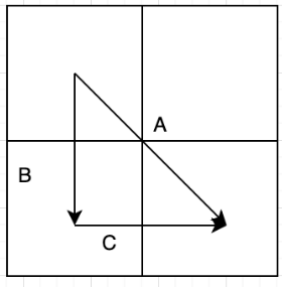
a). Assuming there is no obstacle in the map, and the agent can only move in horizontal (e.g. left or right) and vertical (e.g. up or down) direction. The shortest path will maintain $(x - x_G)$ horizontally moving and $(y - y_G)$ vertically moving, which is $(x - x_G) + (y - y_G)$. As a result, if we take our heuristic as $(x - x_G) + (y - y_G)$, it will never overestimate $h(n)$. Since $a^2 + b^2 \leq (a + b)^2$, we have $(x - x_G) + (y - y_G) > \sqrt{(x - x_G)^2 + (y - y_G)^2}$. And we denote $(x - x_G) + (y - y_G)$ as our new heuristic $h(x, y, x_G, y_G)$. We then have $h(x, y, x_G, y_G) > h_{SLD}(x, y, x_G, y_G)$, which means our new heuristic **dominates** the Straight-Line-Distance heuristic.

Thus, $\boldsymbol{h(x, y, x_G, y_G) = (x - x_G) + (y - y_G)}$.

b).

i). Yes, the **Straight-Line-Distance heuristic is still admissible**. Assume there is no obstacle in the map, the agent can take at least $\sqrt{(x - x_G)^2 + (y - y_G)^2}$ numbers of diagonal movements to reach the goal. And h_{SLD} does not overestimate the heuristic. As a result, it is still admissible.

ii). No, the heuristic from part (a) will **not be admissible** any more. As we have proved from part (a), $(x - x_G) + (y - y_G) > \sqrt{(x - x_G)^2 + (y - y_G)^2}$. And the shortest path will be generated by diagonally moving. Since one diagonally moving can be decomposed into one vertical move and one horizontal move (e.g. A can replace B and C with one cost.). As a result, this heuristic value **overestimates** the shortest path and it is not admissible.



iii).

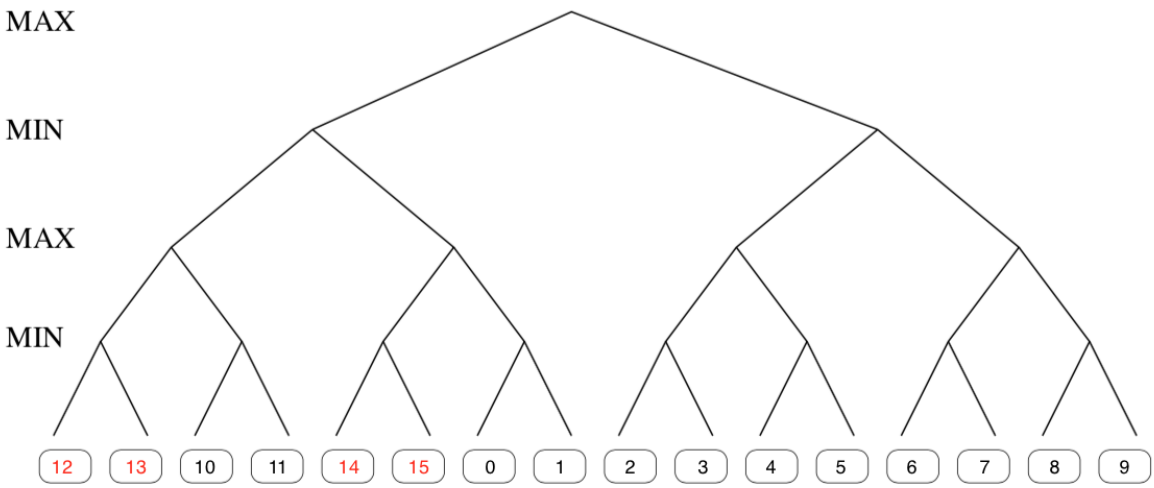
It can be concluded from above question that **each vertical or horizontal movement can be represented by the diagonal movement**. As a result, the goal in location (x_G, y_G) can be achieved with the maximum number of $(x_G - x)$ and $(y_G - y)$ diagonal movements. (e.g. if the goal is located in 7 units down and 5 units left on the start point, there goal will be achieved with 7 diagonal movements). Assume there is no obstacle in the map, the shortest path will be $max(|x_G - x|, |y_G - y|)$, which denotes the maximum value of the absolute value between $x_G - x$ and $y_G - y$.

Thus, $\boldsymbol{h(x, y, x_G, y_G) = max\ (|x_G - x|, |y_G - y|)}$.

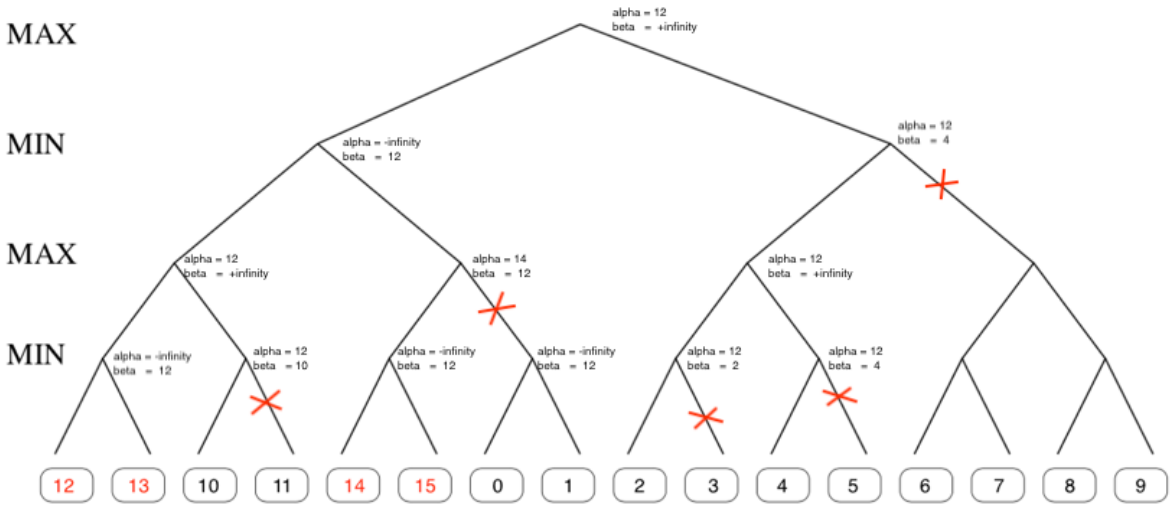
Question 5: Game Trees and Pruning.

a). In a α-β pruning algorithm, a branch will be cut-off when the corresponding α is greater or equal to β. And the β value is updated in a "MIN layer" while the α value is updated in a "MAX layer". As a result, we put more preferable numbers in the left subtree to maximize the pruning.

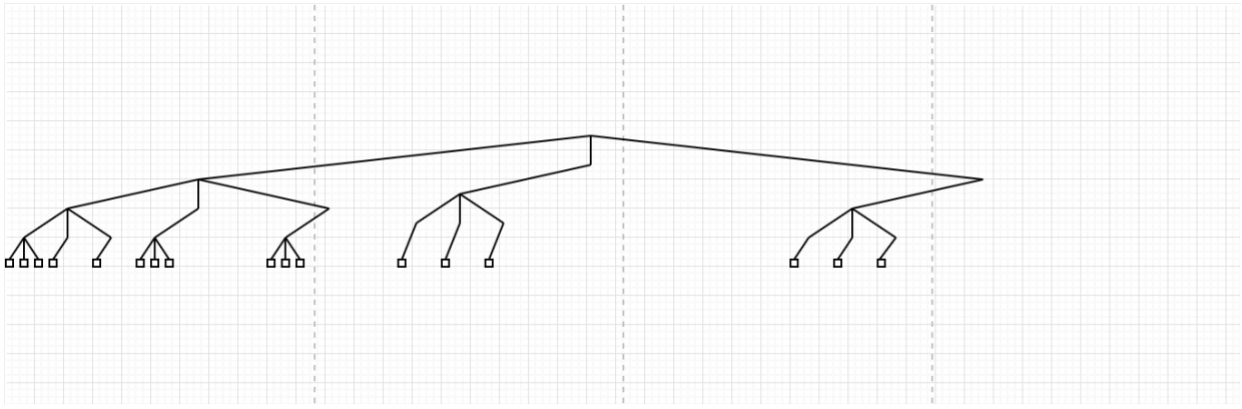
One feasible solution shows as follow. the numbers marked as red are more preferable than the black numbers. As a result, the positions of those red numbers (12, 13, 14, 15) are anchored and fixed. And the location of the black numbers can be changed to generate other feasible answers.



b). The process of the algorithm shows as follow. The condition to cut-off a branch is the corresponding $\alpha \geq \beta$. And the **red cross cuts the branches off.**



c). There are **17 leaves** will be evaluated in the best case (preferable nodes are always in the most left subtree).



d). The time complexity for alpha-beta algorithm is $\boldsymbol{O(b^{m/2})}$**.** The best case of algorithm is all preferable nodes are in the most left subtree. In that case, we can at least cut-off half of the leaves in total. Since the size of remaining leaves is only half of the original size, the take can be completed within $\boldsymbol{O(b^{m/2})}$ (instead of comparing all nodes).