1. Informed Search

1. Traveling Salesman Problem

The state can be represented by cities. Each state can be described as by the following parameter:

Visited, a Boolean variable, indicates if the city has been visited by salesman.

Actions, a list indicate what cities the salesman can visit from this city. (with a cost)

The initial state: salesman is at Waterloo (waterloo is unvisited at beginning of the travel), and other cities are unvisited.

The goal state: salesman is at Waterloo (waterloo will be visited at the end of the travel), and all other cities have been visited.

The neighbour rule:

If there is a route between two cities, these two cities can have the other city in their neighbour list. The cost will be the length of the route between these two cities.

For example, at the beginning, we have these states:

- Barrie:
  - visited: False
  - Neighbours: Guelph (155), Hamilton (149), Toronto (112), Waterloo (173)
- Guelph:
  - visited: False
  - Neighbours: Barrie (155), Hamilton (46), Toronto (95), Waterloo (32)
- Hamilton:
  - visited: False
  - Neighbours: Barrie (149), Guelph (46), Toronto (69), Waterloo (71)
- Toronto:
  - visited: False
  - Neighbours: Barrie (112), Guelph (95), Hamilton (69), Waterloo (113)
- Waterloo:
  - visited: False
  - Neighbours: Barrie (173), Guelph (32), Hamilton (71), Toronto (113)
- Note: The number inside the parentheses is the cost.
- Note: Waterloo is unvisited at the beginning, it will be visited at last.

(B)

Cost function:

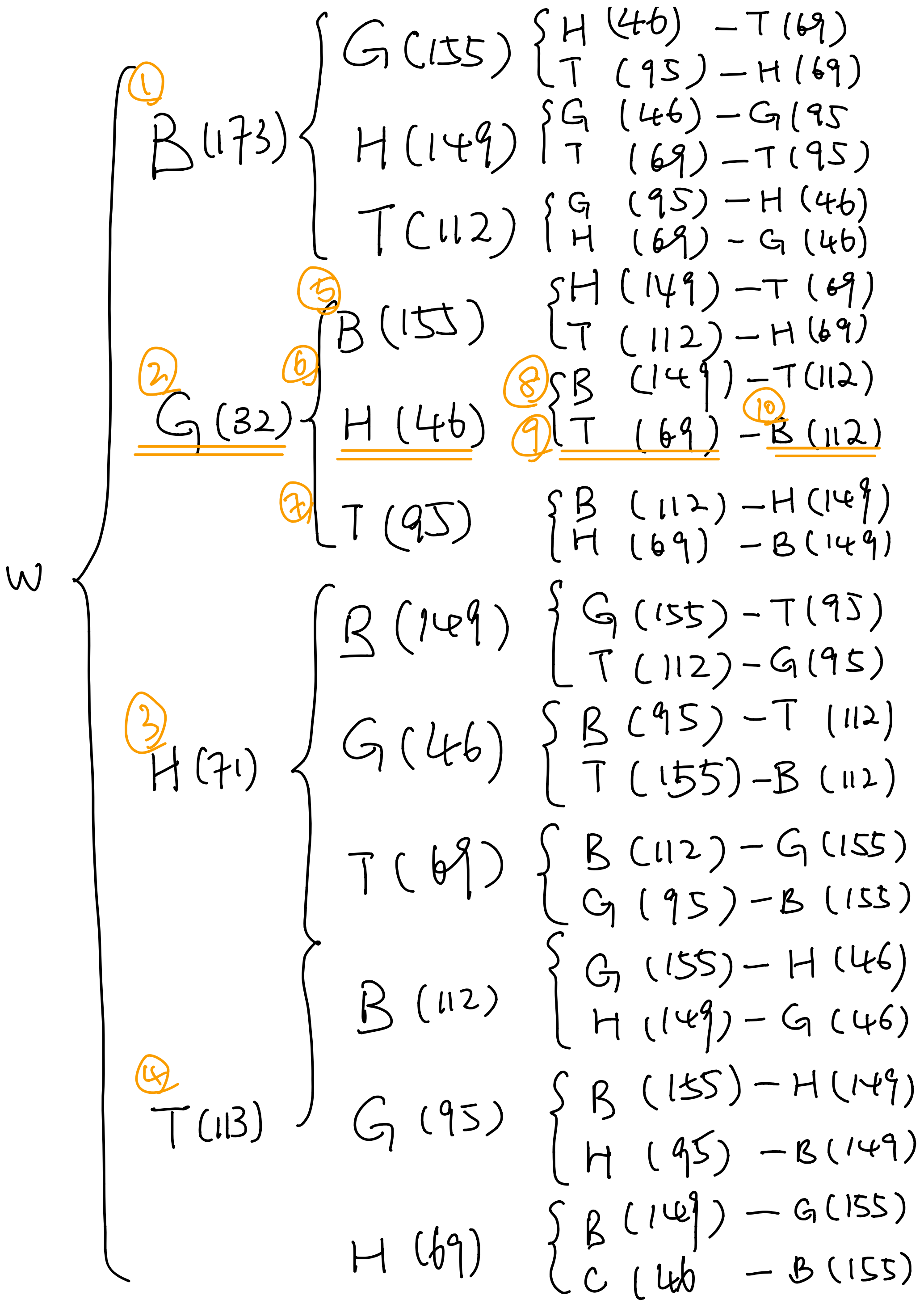
Cost function will be represented as the distance travelled. When the salesman travels to next city, the cost will increase by the cost for travelling to next city (distance between cities).

For example, if the salesman travels from waterloo to Guelph, the total cost will increase from 0 to 32. If the salesman travels to Hamilton next, the total cost will increase from 32 to 78.

(C)

Heuristic function:

- Starting at Waterloo, iteratively select the nearest unvisited city, and return to waterloo when all other cities are visited.
- At the beginning, the salesman is at waterloo and he will travel to Guelph as it is waterloo's nearest unvisited city.
- Then the salesman will visit Hamilton because it is the nearest city to Guelph among Toronto, Hamilton and Barrie.



The total cost will be 32+46+69+112+173 = 432 km

Q2. 8-Puzzle

1. The manhattan distance heuristic performs better than misplaced tiles heuristic. The misplaced tile heuristic only takes the number of misplaced tiles into consideration. As a result, no matter how far a misplaced
   The manhattan distance heuristic also includes how far the tiles are away from their correct positions.
   The manhattan distance heuristic provides more specific information.

	2	3	1	2	3
4	5	6	4	5	6
7	8	1	7		8
	(1)			(2)	

   In example 1, tile 1 is misplaced and in example 2, tile 8 is misplaced. Both statuses will be treated equally by misplaced tiles heuristic.
   In example 1, the misplaced tile 1 is 4 steps from its correct position and in example 2, the misplaced tile 8 is one step from its correction position. The manhattan distance heuristic will evaluate them differently and indicates example 2 is closer to goal configuration.

2. They are monotone.
   * use letter n to indicate after n movement(nth node)
   For the misplaced tile huristic, there are 3 cases:
    a. Move a tile from a non-goal position to non-goal another position, which means the tile is not at the correct position both before and after movement. The value of heuristic function is not changed, but the cost is 1.
       $h(n+1) = h(n)$, which proves that $h(n) - h(n+1) = 0 <= cost(n, n+1)$
    b. Move a tile from a non-goal position to a goal position, which means the one more tile was moved to its correct position.
       The value of heuristic function is decreased by 1, but the cost is 1.
       $h(n+1) = h(n) -1$, which proves that $h(n) - h(n+1) = 1 <= cost(n, n+1)$
    c. Move a tile from a goal position to a non-goal position, which means there is one more misplace tile after movement. Thus, $h(n+1) = h(n) + 1$, and the cost is 1. We can have $h(n)-h(n+1) = -1 <= cost (n, n+1)$

    So that we proved that $h(n) -h(n+1) <= cost (n, n+1)$ for any node n. So that we can have $h(m) - h(n) <= cost(m;,n)$

   For the manhattan distance, move a tile can have two consequences: it is one step closer to its goal position, or one more step away from its target. That is, after a movement the difference of heuristic function's value is,
   $h(n) – h(n+1) = 1$ or $h(n)- h(n+1) = -1$. The cost of a movement is 1.
   Therefore, we proved that $h(n) -h(n+1) <= cost(n, n+1) = 1$
   Finally, for any two statuses(nodes) n and m, we have $h(m) – h(n) < cost(m,n)$

2.Constraint Satisfaction.

1. N-Queens Problem

   1. The variables are the coordinates (row number and column number) of each queen.
   2. Their domain is [1, N], the column number where the queen is placed.
      We can represent the node with a column number and a row number. So there are N variables represented as (x, y) where x is the row number and y is the column number. Since we have a queen in each row, the ith queen will have x = I (1<=i<=N).

   3. The constraints:
      a. Two queens can't be in the same column.
      $$y_i \neq y_j \qquad \forall \quad i \neq j \quad and\, i, j \in [1,2, \ldots, N]$$
      b. Two queens can't be in the same diagonal
      $$|y_i - y_j| \quad \neq \quad |i - j| \qquad \forall \quad i \neq j \quad and\, i, j \in [1,2, \ldots, N]$$

2. N-Samurai Problem

   1. The variables are the coordinates(row number and column number) of each samurai.
   2. Their domain is [1, N], the column number where the queen is placed.
      We can represent the node with a column number and a row number. So there are N variables represented as (x, y) where x is the row number and y is the column number.

      Since we have a queen in each row, the ith queen will have x = I (1<=i<=N).

   3. The constraints:
      a. Two samurais can't be in the same column:
      $$y_i \neq y_j \qquad \forall \quad i \neq j \quad and\, i, j \in [1,2, \ldots, N]$$
      b. Two samurais cannot be in the same M by M block:
      If we say each samurai is in a block. The block can have coordinate (row number and column number from 1 to N/M) The block row number can be calculated as the quotient of row number divided by M. Similarly, the block column number can be calculated as the quotient of column number divided by M.
      For any two samurais, their block coordinates cannot be same. That is,

$$\neg \left[floor\left(\frac{y_i}{M}\right) == floor\left(\frac{y_j}{M}\right) \quad and \;\; floor\left(\frac{x_i}{M}\right) == floor\left(\frac{x_j}{M}\right)\right]$$
$$\forall \quad i \neq j \quad and\, i, j \in [1,2, \ldots, N]$$

3. Sudoku Puzzle
Unary:

1. The variable is the N by N grid.
2. The domains are all possible configuration of the N by N grid.
3. The constraint is that:
    a. Each row of the grid contains the digits from 1 to N, only one time each.
    b. Each column of the grid contains the digits from 1 to N, only one time each.
    c. Each M by M block of the grid contains the digits from 1 to N, only one time each.

Unary:
1. N*N cells, we can represent them as (1,1), (1,2) …...(N, N) where first number is row number and second one is column number, both from 1 to N.
2. Domains: The N positive digits
3. Constraints: there are N constraints:
   AllDifferent {(i, 1), (i, 2), ……, (i,N)} for each i∈{1,2,……,N}
   AllDifferent means that there are no duplicated values in the set.

4. N-Samurai problem and Sudoku.
We can divide the Sudoku problem into n-samurai problems.
1. Firstly, we can solve the N-samurai problem with the grid, but place the digits 1's into the grid instead of N samurais.
2. The output will be a partially filled grid with digits of 1(in total there are N digit 1) in the grid.
3. We can repeat step 1 and 2 to put digits of {2, 3, ……, N} into the grid, each time it will either generate a partially filled grid with filled with another N identical digits or return no possible configuration.
4. When the grid is filled with {1, 2….., N}, N time each. We find a goal configuration of the sudoku.
5. Example: Next page

a. Solve the N-samurai problems but place number 1 into the blanc grid.
Return a grid partially filled with 1 (violate no constraints)

1			
		1	
	1		
			1

b. Solve the N-samurai problems but place number 2 into the output grid we get from setp a.
Return a grid partially filled with 1 and 2

1	2		
		1	2
	1	2	
2			1

c. Solve the N-samurai problems but place number 3 into the output grid we get from step b.
Return a grid partially filled with 1, 2, and 3.

1	2	3	
3		1	2
	1	2	3
2	3		1

d. Solve the N-samurai problems but place number 4 into the output grid we get from step b.
Return a grid partially filled with 1, 2, 3, and 4.

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1