CS486 - Assignment1

Zhaobo Ding

February 3, 2020

1. (a) First we define some variables: denote the number of cities to be n and the starting city to be S. Also, let d_{ij} represent the distance between city i and city j.

   A state of TSP is the path from starting city to current city. As is given in the question, we present the map with a complete, undirected, positive-weighted graph and represent a state with a sequence of node where the following three constraints are met:
   1. The first element of the sequence must be the S.
   2. All non-starting node can appear in the sequence no more than once.
   3. S can be in the sequence twice only when the length of sequence is $(n+1)$ and the later S is the last element of the sequence.

   Since at the beginning we are at starting city, the initial state is the sequence containing only the starting node, which should be written into $\langle S \rangle$.

   To finish this task, we have to traverse the graph without visiting any non-starting node twice. Thus, the goal state is a sequence containing exactly $(n+1)$ nodes.

   For any state S_0, we generate its neighbours depends on its length:
   Case 1: If length of S_0 is less than n. Enumerate all nodes not in S_0, and for each of them, appending to the end of S_0 will generate a new neighbour.
   Case 2: If length of S_0 is n. Append S to the end of S_0 and that is the only neighbour of S_0.

   Cost of a state is the distance from the staring node to ending node in the sequence. In the other word, $cost(\langle n_1, n_2, ..., n_i \rangle) = \Sigma_{j=1}^{i-1} d_{n_j n_{j+1}}$.

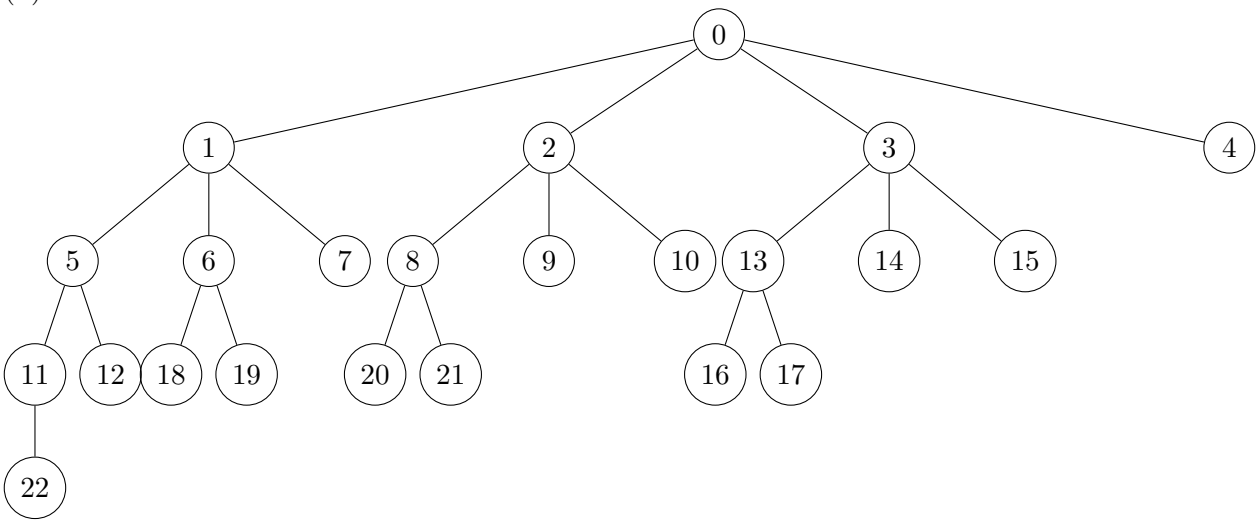
   Use the graph given in question as instance:
   A state is a sequence starting with W, and B, H, G, T can appear only once, and W may appear for the second time only if the length of the sequence is 6 and S appears as the first and last elements. The initial state is $\langle W \rangle$.
   A goal state is a sequence with a length of 6 where S makes up the first and last element and B, H, G, T fills the 4 remaining elements in the middle exactly respectively.

   (b) As is mentioned in (a), $cost(\langle n_1, n_2, ..., n_i \rangle) = \Sigma_{j=1}^{i-1} d_{n_j n_{j+1}}$.
   For example, still, we use the graph in the question as instance.

For initial state $\langle W \rangle$, $cost(\langle W \rangle) = 0$.
For state $\langle W, G, T \rangle$, $cost(\langle W, G, T \rangle) = 32 + 95 = 127$.
For goal state $\langle W, G, H, T, B, W \rangle$, $cost(\langle W, G, H, T, B, W \rangle) = 32 + 46 + 69 + 112 + 173 = 432$.

(c) Let $Remain(s)$ be the nodes not in s, in the other words, $Remain(s)$ is the complement set of s. Then define d_1 be the minimum weight of edge connecting the last node in s and a node in $Remain(s)$. Define d_2 be the sum of $|Remain(s)| - 1$ least weighted edges connecting two nodes from $Remain(s)$. Define d_3 be the minimum weight of edge connecting the a node in $Remain(s)$ and the starting node. Then we have $h(s) = d1 + d2 + d3$.
This function is admissible since the reach a goal state, s must keep searching from current city and go back to the starting city after visiting all remaining cities. Thus we can view the task remaining as three steps: going to an unvisited city from current city, visiting all unvisited cities and go back to starting city. The distance d_1, d_2 and d_3 we defined are the shortest(maybe not possible) distance for each step. Thus $h(s)$ will never overestimate the cost of following search.

(d)



The tree above shows the structure of the search tree and the following table will show details of each node in search tree.

number	expand order	state	cost value	heuristic value	estimate
0	1	$\langle W \rangle$	0	274	274
1	2	$\langle W, G \rangle$	32	298	330
2	3	$\langle W, H \rangle$	71	258	356
3	5	$\langle W, T \rangle$	113	296	409
4		$\langle W, B \rangle$	173	259	432
5	4	$\langle W, G, H \rangle$	78	294	372
6	7	$\langle W, G, T \rangle$	127	289	416
7		$\langle W, G, B \rangle$	187	252	439
8	8	$\langle W, H, T \rangle$	140	282	422
9		$\langle W, H, G \rangle$	117	320	437
10		$\langle W, H, B \rangle$	220	239	459
11	9	$\langle W, G, H, T \rangle$	147	285	432
12		$\langle W, G, H, B \rangle$	227	225	452
13	6	$\langle W, T, H \rangle$	182	233	415
14		$\langle W, T, B \rangle$	225	227	452
15		$\langle W, T, G \rangle$	208	266	474
16		$\langle W, T, H, B \rangle$	331	187	518
17		$\langle W, T, H, G \rangle$	228	328	556
18		$\langle W, G, T, B \rangle$	239	220	459
19		$\langle W, G, T, H \rangle$	196	322	518
20		$\langle W, H, T, B \rangle$	252	187	439
21		$\langle W, H, T, G \rangle$	235	328	563
22		$\langle W, G, H, T, B \rangle$	259	173	432

2. (a) I expect the Manhattan Distance Heuristic to perform better and following are the reasons:
1. As an heuristic function, Manhattan distance estimates the cost more precise than Misplaced tile by simulating the movement of tiles to a certain extent.
2. Further more, Misplaced tile heuristic function only has a range of $[0, 8]$, which is limited and may not help if the number of total steps is huge.
3. In this game, the order of tiles is far more significant comparing to the number of tiles placed correctly since we may need to take many movements just for very few misplacements. Although Manhattan distance heuristic function in some cases has the same issue, it usually performs better on expressing the chaos of our state since in most cases, the far a tile is to its correct location, the more steps we need to take to fix the puzzle.

(b) Misplaced tile heuristic is monotone.
Proof: For any arc $\langle m, n\rangle$, $cost(m, n) = 1$ by definition.
Since only one tile will be moved in a step, the placement of 7 unmoved tiles will not be change, implying that $|h(m) - h(n)| \leq 1$.
Therefore, we have $h(m) - h(n) \leq |h(m) - h(n)| \leq 1 = cost(m, n)$.

Manhattan distance heuristic is also monotone.
Proof: For any arc $\langle m, n\rangle$, $cost(m, n) = 1$ by definition.
Similar to the proof above, only one tile will be moved in one direction, it is obvious that Manhattan normed vector space is a metrics space keeping to triangle inequality, implying that $|h(m)-h(n)| \leq 1$.
Therefore, $h(m) - h(n) \leq |h(m) - h(n)| \leq 1 = cost(m, n)$.

Hence, both of the heuristics are monotone.

3. Variables: (x_i, y_i) for $i = \{1, ..., N\}$ representing the coordinates of queens on the chessboard.
   Domains: $D_i = \{(a, b) : a, b \in \mathbb{N}\ and\ 1 \leq a, b \leq N\}$.
   Constraints: For any distinct $i, j \in \{1, ..., N\}$,

$$x_i \neq x_j$$
$$y_i \neq y_j$$
$$x_i + y_i \neq x_j + y_j$$
$$x_i - y_i \neq x_j - y_j$$

   Formally, concatenate them with $\wedge$, the constraints are

$$(x_i \neq x_j) \wedge (y_i \neq y_j) \wedge (x_i + y_i \neq x_j + y_j) \wedge (x_i - y_i \neq x_j - y_j)$$

   for $i \in \{1, ..., N\}$.

4. Variables: (x_i, y_i) for $i = \{1, ..., N\}$ representing the coordinates of samurai on the chessboard.
   Domains: $D_i = \{(a, b) : a, b \in \mathbb{N}\ and\ 1 \leq a, b \leq N\}$.
   Constraints: For any distinct $i, j \in \{1, ..., N\}$, we have

$$x_i \neq x_j$$
$$y_i \neq y_j$$
$$\lfloor (x_i - 1)/M \rfloor \neq \lfloor (x_j - 1)/M \rfloor \text{ or } \lfloor (y_i - 1)/M \rfloor \neq \lfloor (y_j - 1)/M \rfloor$$

   Formally, concatenate them with $\wedge$, the constraints are

$$(x_i \neq x_j) \wedge (y_i \neq y_j) \wedge ((\lfloor (x_i - 1)/M \rfloor \neq \lfloor (x_j - 1)/M \rfloor) \vee (\lfloor (y_i - 1)/M \rfloor \neq \lfloor (y_j - 1)/M \rfloor))$$

   for $i \in \{1, ..., N\}$.

5. Variables: (n_i, x_i, y_i) for $i = \{1, ..., N^2\}$ representing the numbers to fill and their coordinates on the chessboard.
Domains: $D_i = \{(m, a, b) : m, a, b \in \mathbb{N} \ and \ 1 \leq m, a, b \leq N\}$.
Constraints: For any distinct $i, j \in \{1, ..., N^2\}$,
either $n_i \neq n_j$ or

$$x_i \neq x_j$$
$$y_i \neq y_j$$
$$\lfloor (x_i - 1)/M \rfloor \neq \lfloor (x_j - 1)/M \rfloor \text{ or } \lfloor (y_i - 1)/M \rfloor \neq \lfloor (y_j - 1)/M \rfloor$$

Formally, concatenate them with $\vee$ and $\wedge$, the constraints are

$$(n_i \neq n_j) \vee ((x_i \neq x_j) \wedge (y_i \neq y_j) \wedge ((\lfloor (x_i - 1)/M \rfloor \neq \lfloor (x_j - 1)/M \rfloor) \vee (\lfloor (y_i - 1)/M \rfloor \neq \lfloor (y_j - 1)/M \rfloor)))$$

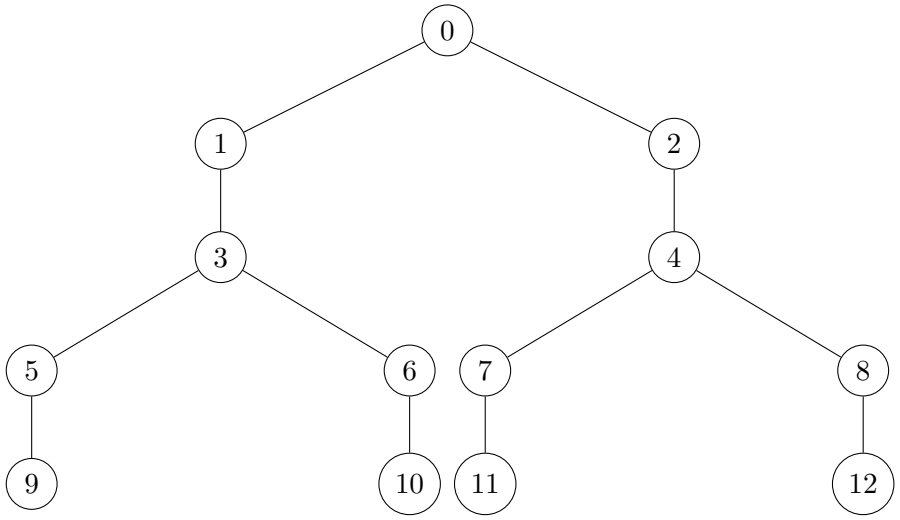
for $i \in \{1, ..., N^2\}$.

6. The input of the CSP-solver is a chessboard with some grids fixed and some blocked, where a fixed grid must be put with a samurai and a blocked grid cannot be put with a samurai and the output is either a failure reporting there is no valid solution or a list of coordinates, each of which is the position to put a samurai on the chessboard based on the restrictions.
   A pseudocode for the searching is given as follow:

Algorithm 1 Sudoku puzzle solver

```
1:  procedure DFS(board, number)
2:      if number > 9 then
3:          Record state                                   ▷ All numbers filled
4:      else
5:          Set all grids on board filled by number to be fixed
6:          Set all grids on board filled by numbers other than number to be blocked
7:          result ← CSP-Solver(board)
8:          if result is Failure then
9:              return
10:         else
11:             for coords ∈ result do
12:                 newBoard ← board                       ▷ Deep copy the board
13:                 Fill number into the newBoard according to the coordinates in coords
14:                 DFS(newBoard, number + 1)              ▷ Search the neighbour
15:
16: init ← initial board
17: DFS(init, 1)                                           ▷ The first number to fill is 1
```

The solution will be all recorded states(recorded in Line 3 in pseudocode) and if no state is recorded, we will get a failure implying that the given chessboard has no valid solution.



Above is the structure of search tree and we will give the states for the nodes in the following page, where obviously node 0 is the initial puzzle and node $9, 10, 11, 12$ are the solutions.

	1		2
	2	1	

0

	1		2
	2	1	
1			
			1

1

	1		2
	2	1	
			1
1			

2

	1		2
	2	1	
1		2	
2			1

3

	1		2
	2	1	
2			1
1		2	

4

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

5

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

6

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

7

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

8

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

9

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

10

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

11

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

12

Since

we are going to find all valid solutions and it is obvious(even from the example above) that the problem may have more than one solutions, we do a deep-first search so that we will perform all necessary expanding while not wasting too much space on storing states in data structure.