COP290 ASSIGNMENT2 SUBTASK2

PARTH GUPTA(2019CS10380)

May 2021

1 The Problem

We are given a 2-D maze with dimensions 50×50. There are some blocked cells and some unblocked cells. We cannot pass through a blocked cell. An unblocked cell is either normal or broken as defined in the Notation section. There are 25 treasures in the grid. We need to move from our start point, take all the treasures and return to our start point. We need to minimize the cost of the path that we take. The cost of a path is defined in the Notation section. We imagine the problem as a Travelling Salesman Problem and use Simulation Annealing.

2 Notation

Any unblocked cell is either normal or broken. We define a weight for each cell C denoted by $w(C)$. If C is blocked then $w(C)$ is infinite, otherwise, if C is a broken cell then $w(C)$ is 10, else $w(C)$ is 1. The cost of a path $C_1, C_2, \ldots, C_n$ is 0 if $n = 1$ and $w(C_2) + w(C_3) + \ldots w(C_n)$ otherwise.

3 Maze Generation

3.1 Basic Structure

We take a 50×50 grid which has all cells blocked initially. We take all the cells which have both x-coordinate and y-coordinate even and unblock them. We need to ensure that these are all connected to each other which we describe in next subsection. Two cells are connected if they share an edge.

3.2 Algorithm

We have used a modified Kruskal algorithm to generate a random maze. We take the unblocked cells as vertices of a graph. Initially, there are no edges. We add edges in this graph. An edge $(v1, v2)$ can be added if the Manhattan distance($\mid v1.x - v2.x \mid + \mid v1.y - v2.y \mid$) is 2 and they have either the same x-coordinate or same y-coordinate. If we add an edge from $v1$ to $v2$, we unblock

the cell that separates $v1$ and $v2$. We need a random maze, not a minimum spanning tree. So, we take edges at random rather than in sorted order(which we do in standard Kruskal's algorithm) and add the edge to the graph if the corresponding vertices are in different components. The maze generated has a tree-like structure, but we want more dense(there should only be a small number of blocked cells) maze for our purpose. We randomly select some blocked cells and unblock them. The blocked cells selected are such that if they are unblocked, they are connected to the tree-like structure. We have now generated our required 2-D maze.

3.3 Random Elements of Maze

We select a unblocked cell at random as the position from where we start(the initial location).
We divide the maze into 25 blocks, each of dimension 10×10. In each block, we select 5 unblocked cells at random. These are our broken cells. Further, in each block we select an unblocked cell randomly and place a treasure there. So, there are 25 treasures and 125 broken cells.

3.4 Shortest Path

We implement a multi-source Dijkstra for finding the shortest distance between all possible pairs of points. Using the shortest distances, we can find the shortest path from any point to any other point. If we had defined the weight of a normal cell as 0, then we could've used 0-1 BFS (it would decrease the time complexity) but it was more suitable to define a positive weight to each cell.

4 Simulation Annealing

First we generate a random order to take the treasure. Let the sequence generated be S and the associated cost be C. We define a variable T(for temperature) which is initially 1.0 and another variable $k = 50$. Then we run a loop with 50 iterations. In each iteration, we multiply T by 0.9 . After that, we run another loop of 250 iterations inside the first loop. In each iteration of the inner loop, we take 2 elements of S randomly and swap them. This gives us another sequence S' with cost C'. If $C' < C$ then we do $S = S'$. Otherwise, first we generate a random floating-point number r. If $e^{-\frac{C'-C}{k \times T}} > r$ then we do $S = S'$ else we do nothing and move onto the next iteration of the inner loop. When we exit the outer loop, we have obtained the sequence S and the cost C generated by the algorithm. We provide the pseudo code below

5 A Deterministic Approach

The most straightforward approach is taking all the possible permutations in which we can take the treasures. But we would need to calculate the cost for 25! different sequences.
There is another approach which uses bitmasks and dp. Suppose we have reached a position where we have taken some treasures. We can store this as a single 25 bit integer. The i^{th} bit of the integer is 1 if we have already taken the i^{th} treasure(give any numbering to the treasures). We also need to know the treasure that we have taken last among all the treasures taken. So, for any position, we can define a dp state dp[i][j], where i represents the treasures taken and j is the number of the treasure taken last.
Our recursion is:
dp[i|(1<<s)][s]=min(dp[i|(1<<s)][s], dp[i][j]+distance(s,j)) for all treasures s not taken currently.
The base case is dp[0][source]=0 and dp[0][others]=infinite.
Our final answer is min(dp[$2^{25}-1$][0]+dis[0][source], dp[$2^{25}-1$][1]+dis[1][source], ..., dp[$2^{25}-1$][24]+dis[24][source])