Son Vo

Professor Tannaz R.Damavandi

May 8th 2020

Detailed Report

Task #1: **Use backtracking technique to solve a given Sudoku puzzle**

I. Approach to solve Sudoku:

Step 1: First, we find the empty space in the Sudoku.

Step 2: For each empty space we will try place all number from 1 to 9

Step 3: Check if the number is valid at the empty cell, check row, column and sub-gird

Step 4: If the number is valid, insert the number to current spot, then go back to step 1

Step 5 if the number is invalid, reset the square and go back to previous step

II. Difficulties:

How to find which sub gird that current zero belong too is one challenge that I've encountered.

The solution is pretty simple. We get the index of row and column of the current box by use floor division for current index i and j. As a result, the new index will be range from 0 to 2 which indicates which box the empty space belong to.

III. Time complexity: $O(n^2)$

Task #2: use best first search with branch and bound algorithm to solve 0/1 knapsack

I. Approach to solve 0/1 knapsack problem:

Step 1: we initialize the root and add the root to priority queue, set max profit = 0, create two node u and v.

Step 2: if the priority queue has node in it. pop the node with best bound from priority queue and store it in v node

Step 3: if the bound of v node is bigger than max profit then go to next step

Step 4: set u to the child that includes the next item.

Step 5: if u weight is less than capacity and u profit is bigger than max profit then update the max profit = u profit

Step 6: get the bound of u node.

Step 7: if the bound of u is greater or equal to the max profit then add the u node to priority queue

Step 8: set u to child that does not include the next item.

Step 9: get the bound of u node

Step 10: if the bound of u is greater or equal to the max profit then add the u node to priority queue. Then go back to step 2.

Step 11: when there isn't any nodes priority queue then we have the max profit.

II. Difficulties:

The nodes in the priority queue were all the same values because I used the same u node for left and right children of root. As a result, when I updated the value for u, the node in the priority queue also changed. It took me a lot of times to realized and my program finally worked.

Another difficulty was that I could not figure out how to get the optimal set. Due to time limitation, I had to move on the next task.

III. Time complexity: $O(2^n)$

Task #3: Use the best first search with branch and bound algorithm to solve Traveling salesman

I. Approach to solve Traveling salesman problem

Step 1: we initialize the root and add the root to priority queue

Step 2: if the priority queue has node in it. pop the node with lowest bound then store into v node.

Step 3: if v bound is smaller than the min length. Then set u level to child of v

Step 4: for all I such that i < n and i is not in v path set u path = v path and add i to the end of u path

Step 5: if u level == n -2 put the last remain vertex to path then put the first vertex at last. Also check if the length of u < min length, If its is smaller than min length then update the min length, shortest length and optimal tour

Step 6: u level != n -2, set find u bound, if u bound < min length then add u to the priority queue

Step 7: sort priority queue base on the bound in increasing order then back to step 2.

Step 8: when there isn't any nodes priority queue then we have the shortest length and optimal tour.

II. Time complexity O(n!)