COP290: DefusoBot Simulation

Mustafa Chasmai | Tamajit Banerjee
2019CS10341 | 2019CS10408

May 22, 2021

Introduction

As part of the second task of the course COP290, we have made a simulation to visualise a particular problem statement. We propose various methods to solve this problem and demonstrate the same as a running simulation.

In this report, we describe our problem statement in detail and discuss about the algorithms we considered to solve the problem. Different algorithms have different characteristics, and based on particular conditions imposed (optimize at all cost, optimize but with a limited complexity etc.), different algorithms may be the optimal choice.

1 Problem Formulation:

1.1 The Cost-Constrained Travelling Salesman Problem (CCTSP)

This is a variation of the Travelling Salesman problem, where instead of optimising the cost for visiting all nodes, we optimise the number of nodes we can visit with a limited cost.

Like the Traveling Salesman Problem, the Cost-Constrained Traveling Salesman Problem [5] [2] [4] can be formulated as an optimization problem. We define the optimization problem as follows:

Given a set of nodes 1,2,..,n, a non-negative cost matrix C = $[c_{ij}]$, positive values $v_l, v_2, ..., v_n$, and a budget B;

$$\underset{\pi(m)}{\text{Maximize}} \quad \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{v}_{\pi(\mathrm{i})}$$

$$\text{subject to} \quad \sum_{i=1}^{m-1} c_{\pi(i),\pi(i+1)} + c_{\pi(m),\pi(1)} \preceq B,\ m \in [1,n]$$

In this basic formulation of the Cost-Constrained Traveling Salesman Problem, we require the subtour taken to reach all locations to be a closed loop starting and ending at a specified node. The nodes may have different values. Thus, a node that could have been reached may not be visited if returning back from the node would not be feasible within the budget.

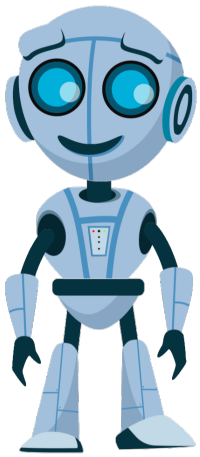
1.2 Problem Statement

We consider a particular situation where the CCTSP becomes very relevant. Let us consider a war-ridden region, with excessive use of weapons and in particular bombs. In such regions, there may be bombs or mines located randomly throughout the location, taking one wrong step may lead to the loss of a person's life. To return any sense of stability in such a region, it is imperative to locate and defuse all such bombs/mines. Thus, a bomb defusal specialist would have to risk his/her life while defusing bombs. For quite some time, robots are steadily replacing humans for this risky job. In lieu of this, we simulate this situation, and demonstrate how the robot may behave and maximise the lives it saves.

Lets consider this real-life problem in detail. We have a bomb-defusal robot, with a limited battery capacity. The robot has to try to defuse as many bombs as it can, and then return to its base for recharging. Again, the robot may chose to prioritise the bombs, with parameters like the blast capacity, expected population that may be affected and expected time remaining for blast.

We simulate this real-world scenario. The region where the bombs are placed can be approximated as a general maze (or graph), where an edge between two nodes would mean a road between two locations. The bombs are placed at random positions in the maze and the robot starts at its base (some random position in the maze). Each bomb has an associated value with it (based on the expected number of lives saved by defusing it), and the robot has to maximise the number of lives it saves before it has to return to the base for recharging. The robots uses up some of its battery life going from one location to the other and them some more when it defuses a bomb. Thus, visiting and defusing each bomb has an associated battery-life cost with it.

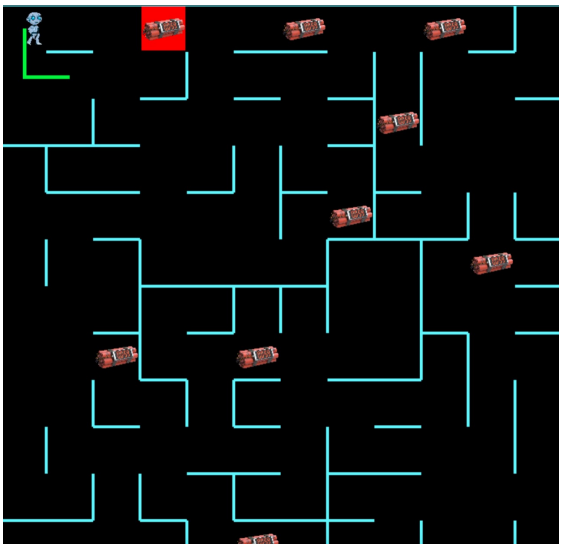
Thus, this translates to a CCSTP where the robot has to maximise the number of lives it saves (value), with a constrain on battery-life it has (cost). We consider different algorithms that the robot may use to achieve this task, and try to find and algorithm that will allow it to save a maximum number of lives.



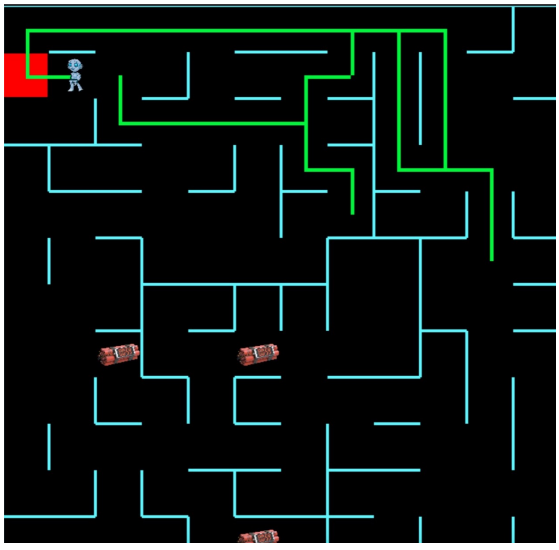
2 Algorithms

2.1 Exact Algorithm | Dynamic Programming based (using bit-masking)

This exact algorithm [5] searches for the most optimised solution. It has a higher complexity than other heuristic-based approaches, but it guarantees that the optimal solution is found. In this algorithm the problem is viewed as a sequential decision problem. At each stage of the algorithm, an additional node (bomb) is visited, and then which node to visit next is decided.



The Maze Simulation Starting



Completing the tour using DP

In the first figure, the droid has just left from the base. The bomb site with a red background is where it is planning to go next. In the second figure, the robot has completed the tour and is returning to the base. We had kept the budget B to be 40, and the droid could not defuse all the bombs in this budget.

Algorithm 1: Dynamic Programming based (using bit-masking)

```
dpCost[2^n][n] ← intMax;
path ← empty vector of integers;
parent[2^n][n] ← 0;
for mask ← 0 to 2^n - 1 do
    if mask = 0 then
        for bomb ← 0 to n do
            dpCost[mask | (2^bomb)][bomb] ← cost[n][bomb];
            parent[mask | (2^bomb)][bomb] ← n;
        end
    else
        for bomb ← 0 to n do
            if NOT ( mask = 0 & 2^bomb ) then
                for pos ← 0 to n do
                    if dpCost[mask | 2^bomb][bomb] > dpcost[mask][pos] + cost[pos][bomb] then
                        if mask = 0 & 2^pos then
                            dpCost[mask | 2^bomb][bomb] ← cost[n][bomb];
                            parent[mask | 2^bomb][bomb] ← n;
                        else
                    else
                        Do Nothing
                    end
                end
                else
                | Do Nothing
                end
                dpCost[mask | (2^bomb)][bomb] ← cost[n][bomb];
                parent[mask | (2^bomb)][bomb] ← n;
            end
        end
    end
end
```

2.1.1 Analysis

Time Complexity : Here we iterate over all the masks possible and also all the last vertex visited in that mask and we need O(n) time to calculate each dp value and there are in total 2^n*n states. So the total time complexity is O($n^2 * 2^n$) .

Advantages: Gives the optimal price value every time.

Disadvantages: Takes exponential time to compute so infeasible in real life scenario.

2.2 Basic Heuristic

1. Here, instead of searching for the best possible tour when we are starting from a node , we try to search the best possible next node which we can visit. The heuristic [3] [1] decides which node to go next.

2. In this heuristic, we use a modified value, calculated as:

$$agg_i = X * v_i + \sum_{j \in S, j \neq i} v_j e^{-\mu_1 c_{ij}} + (1/Y) * \sum_{j \notin S} v_j e^{-\mu_2 c_{ij}}$$

   where S is the set of remaining bombs to be defused

3. Based on these values, at each step of the algorithm, the next node to visit is decided. The node selection criteria is:

$$\underset{i}{\text{Maximize}} \quad \frac{agg_i}{travelCost_{icur}}$$

   where $travelCost_{i,j}$ means the cost to travel from current node j to node i and the current node is denoted by cur.

4. We first consider a basic heuristic of always going to the nodes in sorted decreasing order of their agg_i values. We pre-compute the distances from each cell in the maze to each other cell, and use these to pre-compute these aggregate values. Again, before deciding on the node to visit, we should first check whether it would be feasible, i.e. going back to the starting position is inside the budget.

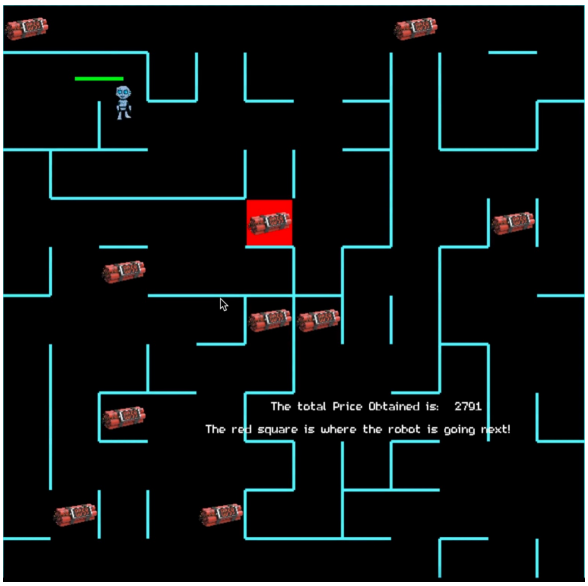
2.2.1 Analysis

(a) **Time Complexity** : We have to do at most n iterations to add all of them in the tour. In each iteration we need to update their aggregate values due to the currently added vertex. So O(1) for each vertex and so n updates in each iteration and selecting the best out of them which takes O(n*logn) . So it takes O(n^2*logn). We terminate if we are not able to add a node in that iteration.

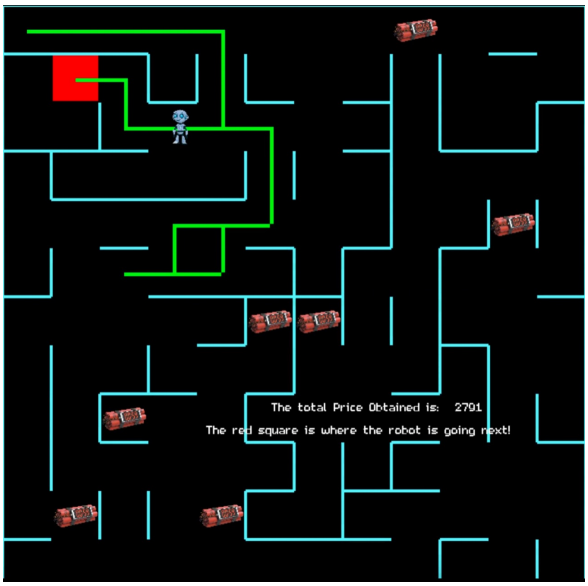
(b) **Advantages**: We can easily identify the clusters and so when the robot visits a cluster, it can easily reach to it's neighbouring nodes with lower costs. It prioritises the ones having nodes with high price value as close neighbours. Thus, prioritising clusters would eventually optimise the search.

(c) **Disadvantages**: It Does not easily identify clusters far away from current node. It does not give priority to a node having quite high price value but no nearby neighbours.

.



The Maze Simulation Starting



Completing the tour using Basic Heuristic

In the first figure, the droid has just left from the base. The bomb site with a red background is where it is planning to go next. In the second figure, the robot has completed the tour and is returning to the base. We had kept the budget B to be 40, and the droid could not defuse all the bombs in this budget.

2.3 Advanced Heuristics

1. For the first **N** nodes, we use a simple cost/profit heuristic to decide which node to visit in each stage. The node selection criteria for the first **N** nodes is:

$$\underset{i}{\text{Maximize}} \quad \frac{v_i}{travelCost_{i,0}}$$

   where $travelCost_{i,j}$ means the cost to travel from current node j to node i. The number of such nodes, **N** is a parameter that can be tuned.

2. For the remaining n - **N** nodes, we define:

$$agg_i = X * v_i + \sum_{j \in S, j \neq i} v_j e^{-\mu c_{ij}} + (1/Y) * \sum_{j \notin S} v_j e^{-\mu c_{ij}}$$

   where S = set of remaining bombs to be defused. The terms **X** and **Y** here are parameters that can be tuned.

3. Based on these pre-computed aggregate values, we select the next node to visit in each stage. The node selection criteria is:

$$\underset{i}{\text{Maximize}} \quad \frac{agg_i}{insertionCost_i}$$

   where $insertionCost_i$ means the cost to insert that particular node in the current tour.

4. Initially we start by selecting points by sorting them by the first maximise criteria. We select the best possible 3 points out of the first 6 points with highest such value.

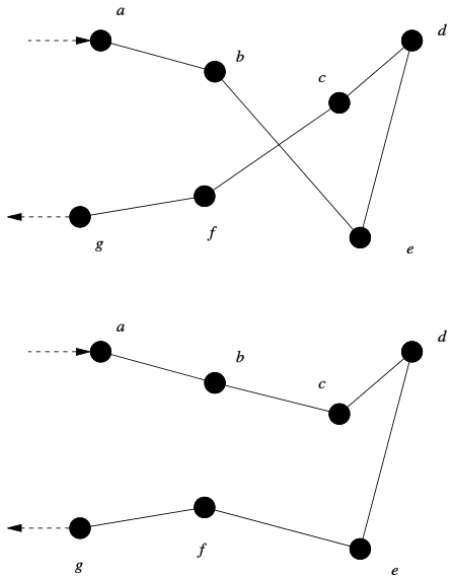
5. if selecting 3 points is not possible this means we do not have much time in hand so we can use the basic heuristic.

6. Otherwise we start the next maximisation algorithm starting from a tour of the starting point and the 3 points chosen.

7. For each bomb not in the tour, find its learning measures. Then chose the bomb with the highest value, and insert it using insertion algorithm explained after this.

8. Then we apply 2-opt algorithm after each iterations and we do an iterations on each of the edges.

9. Repeat step 2 until no more bombs can be defused in the given time constraint else terminate.



The 2-opt algorithm

2.3.1 Analysis

1. **Time Complexity** :
   (a) At first we create the short tour and it takes O(n*logn) due to sorting.
   (b) We have to do at most n iterations to add all of them in the tour. In each iteration we need to update their aggregate values due to the currently added vertex. So 1 update to each vertex in each iterations for changing the aggregate value.
   (c) Then we have to find the cost for insertion cost for each vertex. We can do this by iterating over the edges and trying to add the new node by breaking the edge and taking the minimum out of them. There are O(n) edges and O(n) nodes so the time complexity for insertion is O(n^2).
   (d) We also apply 2-opt algorithm in each iteration . In this we select an edge and try to iterate over all the edges and see if they intersect if they do then we can manipulate them as shown in the diagram and get a better cost to reach there. So there are O(n) edges so it takes O(n^2) time.
   (e) The total complexity is O(n^3) as we run 2-opt algorithm and insertion cost update algorithm.
2. **Advantages**: We can easily identify the clusters and also the nodes with big prices far away which will eventually increase the answer. When the robot visits a cluster, it can easily reach to it's neighbouring nodes in lower costs. It prioritises the ones having nodes with high price value as close neighbours. Thus, prioritising clusters would eventually optimise the search.
3. **Disadvantages**: It can easily identify clusters far away from starting node as we are selecting the initial nodes on the basis of price. So it helps in the case when we have relatively more time to explore the far away nodes.

3 Simulation

3.1 The Maze

1. The maze is represented as a graph, with each cell of the maze as a node. By removing walls between cells, we create edges between nodes. To create a general connected graph, we remove walls in the form of a spanning tree tour and then remove some extra walls so that we can get a denser graph, which can realise more realistic scenarios.

2. The maze satisfies the Triangle Inequality: $\forall(i,j,k):\ d(i,j) \leq d(i,k) + d(k,j)$, and thus, is a **metric space**. This is relevant, and is used as an assumption in one of our algorithms.

3.2 The Robot

1. The robot starts at the center of some cell in the maze. This cell is considered the base of the robot, and it has to return to this cell at the end (If the budget does not allow it to go to a new position and return, it will not go to that position).

2. The robot can move using the two algorithms we have discussed above. In one mode, it uses dynamic programming and considers every possible tour before selecting the best one possible. In the second heuristic-based algorithm, it uses the heuristics we defined in earlier sections to find the path, and then follows it.

3.3 The Display

1. As the robot moves around, it sometimes becomes difficult to analyse and evaluate the performance of the algorithm. To make this easier, we display various information on the screen of the simulation.

2. At every step of a path, the robot decides to travel to some target maze cell. This target cell is shown with a red colored square on top of it. Along with this, the total profit gained by the robot is displayed. We observed that the heuristic based algorithms tend to give lower profits than the dynamic programming based one.

3.4 The simulation itself

1. All functionalities discussed in the earlier subsections are implemented in SDL and C++. The code source files along with the instructions of use can be found in the **git repository** here.

2. Two particular examples where the difference between the two algorithms we consider can be seen in the **video** here. In the first example, it can be seen that the two algorithms give the exam same profit, and that the actual paths differ only in one or two places. In the second example (second half of the video), the two algorithms give very different paths, and yet their costs were very close. Upon closer inspection, we observed that one of the nodes had slightly higher profit than another, but was also a bit costlier than the other (still within budget). The profit/cost used in the heuristic, thus, was greater for the latter, and the heuristic-based algorithm gave preference to that node over the other.

4 Comparison with general TSP

4.1 Similarities

1. CCTSP(recognition) is NP-complete for general graphs.
2. CCTSP(optimization) is NP-hard for general graphs.
3. There is no fully polynomial approximation scheme for CCTSP unless P = NP.

4.2 Differences

1. For metric graphs , TSP has some Polynomial time algorithm that gives a tour of cost < (constant) * optimal cost
2. The nearest neighbour algorithm also gives an lower bound of the optimal solution possible.
3. These algorithms cannot be completely applied to CCTSP because of its constraint on time and the inclusion of the price parameter.

4.2.1 Example of a general TSP algorithm workin in polynomial time

Algorithm 2: A Polynomial time algorithm that gives a tour of cost < 2 x optimal cost

Input: G (the TSP graph being considered);

1 T ← Minimum Spanning Tree of G;

2 π ← DFS(T);

3 $\pi*$ ← First Occurrences of nodes in π

Proof:

1. **Claim 1: Cost(T) $\leq$ Optimal Cost**

   This can be proved by contradiction. If this was not true, i.e., say Cost(T) > Optimal Cost, then the Optimal tour cannot be a spanning tree. Thus, it either contains a cycle or is not spanning, both of which lead us to a contradiction.

2. **Claim 2: Cost(π) = 2 * Cost(T)**

   Since π is obtained by performing a DFS on T, it has to contain each of the edges exactly twice. Thus, its cost also has to be twice that of T.

3. **Claim 3: Cost($\pi*$) < Cost(π)**

   $\pi*$ is obtained by removing some nodes and their corresponding edges from π. So, this relation comes naturally.

From Claims 1, 2 and 3, it follows that Cost($\pi*$) < optimal cost.

5 Conclusion

1. The general CCTSP problem is challenging because it depends on two variables, the profit and the cost. This differs from normal TSP, where only cost has to be minimised.
2. We have tried to implement and explore most of the general heuristics used in TSP and modify them to to include the price as a constraint in those heuristics.
3. We could not achieve an upper bound for the optimal profits using the results we get from our heuristics, but we have tested our algorithms on several randomised test cases and have been able to set the tunable parameters to some values which will give results close to the optimised one even for general test cases.
4. We believe that limiting the maximum value of the profit obtained by visiting individual nodes and also some specific graph structure, will allow us to give upper bounds on the optimal solution based on the best approximation we can make using a polynomial time algorithms.
5. We can also try to improve the heuristics by concentrating on other factors as well, for example, based on reachability from the current node and using the information about the remain cost in the budget.

6 Links to View Simulation

1. The source code for the simulation can be found at the git repository here. (https://github.com/tamajit-banerjee/COP290_Task_2_Subtask_2)
2. Demo containing two examples of the simulation running: here (https://drive.google.com/file/d/14gMiHiBEFFTxv-cjiOJDQr8yoxb2sQjZ/view?usp=sharing)

References

[1] Carnegie Mellon University Lecture Notes. "The An-Kleinberg-Shmoys Algorithm for the Traveling Salesman Path Problem". In: (). URL: https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture19.pdf.

[2] Samir Maity, Arindam Roy, and Manoranjan Maiti. "Rough Genetic Algorithm for Constrained Solid TSP with Interval Valued Costs and Times". In: *Fuzzy Information and Engineering* 10.2 (2018), pp. 145–177. DOI: 10.1080/16168658.2018.1517972. eprint: https://doi.org/10.1080/16168658.2018.1517972. URL: https://doi.org/10.1080/16168658.2018.1517972.

[3] Christian Nilsson. "Heuristics for the Traveling Salesman Problem". In: (Jan. 2003).

[4] Stephanie Tauber Professor Lenore Cowen. "The Travelling Salesman Problem (TSP)". In: (). URL: http://www.cs.tufts.edu/~cowen/advanced/2002/adv-lect3.pdf.

[5] P.R. Sokkappa. *The cost-constrained traveling salesman problem*. Oct. 1990. URL: https://digital.library.unt.edu/ark:/67531/metadc1113789/.