Reading By Visualisation Derivation

Harsh Aggarwal, Pranjal Aggarwal

1 Introduction

With advent of modern technology, once a thing of future, is now becoming a common reality. However AI agents employed in these systems have to face various challanges, as the cognitive abilities of AI is far less than that of humans. One such task is that of robot cleaning. The robot systems employed in this task have to sense the enviornment and take decisions without harming the humans or other objects. At the same time these systems should be efficient enough to complete their task in a limited time and also in limited amount of energy. In this report we study this problem of efficiency, and discuss various literature, which have tried to solve a problem similar to this. We use them and build upon a new heuristic based approach.

2 Problem Statement

Suppose we have an indoor building with n rooms, which need to be cleaned. Each room may have different sizes and floorplans, and different amount of dust in them. The task of the robot is to clean all the rooms in the building in a limited amount of time.

[This can be due to both limited battery capacity of robot or urgency to complete the tasl]

At the same the robot has to maximise the amount of cumulative amount of dust cleaned in all rooms, while providing a minimum in each of the room. The problem can be modelled as a two stage problem:

- In the first stage, we find the best path for the robot to reach all the rooms. This is a Travelling Salesman Problem we propose Steiner TSP Algorithm for this.
- In the second stage, we allot a certain time to each room based on their size, and the robot has to find a path which maximizes the dust cleaned, while at the same time is within the alloted budget of time.

Both of these stages can be modelled as a path finding in graph problem, where in first stage the pathway connecting rooms is equivalent to edges and the rooms as vertices. For the second stage we divide a room into a grid of n*n cells, each either being occupied by an object, or having certain amount of dust. This grid can be considered as graph, with the unoccupied cells as the vertices and the amount of dust in them as their vertex value. Since it will take slightly more time for robot to clean cells with larger dust amount, the edge lengths will be dependent on the values of both the connected vertices:

$$C_{i,j} = k + f(V_i, V_j) \tag{1}$$

Where k is a constant value dependent on the speed of robot, and f is an arbitary function defined on the ordered values of two vertices.

3 Related Work

The problem at hand can be modelled as a variant of TSP asknown Profit-TSP. In this problem, the Salesman has to maximise its profits however in a fixed finite amount of time. This has been shown to an NP-Hard Problem [1].

4 Solution

In this section we will discuss two possible heuristic solutions to our problem.

4.1 Centre of Gravity Based Algorithm

4.2 Multi-Stage Algorithm

While the previous algorithm requires an Euclidean cost function, in this algorithm we will model a genric enough cost function and solve the problem.

We divide the solution to the problem in multiple stages each performing a different operation. All these stages are iterated over for multiple rounds, till convergence criteria is not met. We use 4 different hyperparameters:-

4.2.1 Stage I: Insertion Stage

At the start of stage we assume that we have a path $\{L(i) : 0 < i \leq k\}$, where k is the path length, and L(i) denotes the i^{th} vertex in the path. Since we have to start and end at the same position, $L(1), L(2) = 1$. We define two sets Ψ and Ω , which contains the vertices in current path and and vertices not in current path respectively. Now we first decide, a vertex to insert, lets call it j. Now we need to find a point in the path where we can insert the given vertex, and which gives maximum increase in profit while minimising the cost. Then point of insertion can be found by solving the following Optimisation:

$$\Theta_j min_{i=1,\ldots,k}(C_{V(i),j} + C_{j,V(i+1)} - C_{V(i),V(i-1)}) \tag{2}$$

Where Θ_j is the calculated cost of given vertex for insertion. Also note that in 2 we only need to iterate over only neigbours of vertex j, which in our case can be a maximum of 8, and therefore step 1 can be done in O(1) time.

Now we iterate over all the sutiable vertices for insertions, i.e and calculate the corresponding Θ_j. Now the best vertex is the one which give the maximum profit while with minimum cost i.e we try to take that vertex which has the highest profit to cost ratio. Therefore we have to solve the following equation:

$$max_{j\in\Omega}\frac{P(j)}{C(j)}) \tag{3}$$

However this would give high weightage to high density vertices therefore we add an additional hyperparameter λ_1 in 3, and modify it as:

$$max_{j\in\Omega}\frac{P(j)^{\lambda_1}}{C(j)} \tag{4}$$

We hypothesize that $\lambda_1 < 1$. This is due the fact that a minimum time is always required for robot to cross a cell, and thus by adding this additional hyperparameter, we ensure that portions of small dust are not leftover.

We continue this process of insertion till the cost doesnt exceed $\lambda_{2,1} * \beta$, where $\lambda_{2,k}$ is another hyperparameter whose value varies in different rounds, and always increasing in subsequent rounds.

If addition of any vertex increases the cost beyound alloted capacity, we omit it and move on to the next stage.

As we see that runtime complexity of this step is $O(n^2)$ in this case, wehere n is the final path length achieved after performin this stage. Note that its not $O(n^3)$ due to limited number of neighbours a vertex can have in the maze like structure we have used.

4.2.2 Stage 2: k-Opt

In this stage we apply the famous k-Opt Startegy [2] increase the total profit while being in the budget. We start with applying 2-opt Optimisation, and if the final profit is more than $\{\lambda_3, \lambda_3 > 1\}$ times the previous score, we move on to higher order k-opts. However if increase in budget is observed without satisfying the previous equation, we go back to stage-1 since further insertions may have scope for improvement. However if no increase in profit is improved, we move onto the next stage. The worst case complexity of this strategy on a complete graph is $O(n^2)$, however as in previous stage, since we have a maximum of 8 connexting edges from a vertex, the time complexity is just $O(n)$ for a single pass of this stage.

4.2.3 Stage 3: Deletion

In this stage we try and delete vertices, that may not be useful for inclusion in path. Let S_{final} be the new profit after deleting a vertex v from path P. Then we define a metric:

$$\xi(j) = \frac{S_{initial} - S_{final}}{P(j)} \tag{5}$$

where j is the vertex removed from the path

Then we find the vertex $j \in P$ such that value of ξ is minimised. What this means is that deletion of j gives the lowest per unit price loss. After this step we check for the convergence criteria which terminates when the new overall profit P_{ov} is atleast λ_4 times the overall profit in last round $P_{ov,last}$. If the convergence criteria is not met we move back to stage 1 for further insertions and repeat the steps. Again the time complexity of this stage is $O(n)$ since we can delete only n points in the path, and deleteion of each point takes only $O(1)$ time

4.2.4 Conclusion

As we have shown the overall complexity of each stage is not more than $O(n^2)$ and therefore each round of algorithm is fast enough. Also results in literature show that similar algorithms are able to achieve fast enough convergence, that they can be used in real time embedded systems, with minimum energy costs. [**?**] Simulation data suggests that the robot was able to find a very good path in very less time, thus showing the effectiveness of this algorithm.

4.2.5 Final Algorithm

Algorithm 1: Optimisation Algorithm

while *not converged* **do**

1. Stage 1: Given a graph $G = (V, E)$ perform insertions based on 4 till the cost doesnt exceed the specified value;
2. Stage 2: Apply k-opt strategy, and if improvements are good enough go back to stage 1. Else go to Stage 3 ;
3. Stage 3: Perform Deletions of vertex in path in accordance with 5;

end

5 Conclusion

References

[1] PlaceHolder. Placeholder, PlaceHolder.

[2] PlaceHolder. Placeholder, PlaceHolder.