Optimisation of path for Ganesh Procession

Nishant Raj
Department of Computer Science and Engineering,
National Institute of Technology,
Surathkal, India

R. Vishwanath
Department of Computer Science and Engineering,
National Institute of Technology,
Surathkal, India

Anubhav Jain
Department of Computer Science and Engineering,
National Institute of Technology,
Surathkal, India

K. Chandrasekaran
Professor
Department of Computer Science and Engineering,
National Institute of Technology,
Surathkal, India

***Abstract*—Ganesha Chavithi is one of the most important religious festivals conducted through out India which includes Ganesha procession followed by Ganesha immersion. This procession leads to high traffic congestion which in-turn leads to disruption in routine activities of the citizens. This paper discusses about the various approaches that can be applied so that the traffic congestion can be minimized and thus resulting in comfort of common people. In this paper, we proposed a novel algorithm based on shortest distance for clustering and optimized MST algorithm for finding optimal path for each cluster of Ganeshas to it's Lake. and have also shown the results as implemented on google maps.**

KEY WORDS: Dijkstra's, Minimum Spanning Tree (MST), Clustering.

I. Introduction

Prerequisite Context:

Ganesha Chaturthi is one of the most important religious festivals that happen in India. The festival duration is 10 days. The festival is recognized by placing Ganesha idols privately in homes, publicly in temporary stages called as Mandaps which are distributed throughout the city and all over the country. On the eleventh day of festivals, all the installed Ganesha idols are taken for the procession through the city and procession is followed by immersion of idol.

Problem Description:

Ganesha idols are placed in various locations of the city. The number of Ganesha idols are increasing year by year and along with that the number of Ganesha idols depend also on the area of the city. The number is huge in big cities like Hyderabad, Mumbai, Bangalore etc. In Mumbai itself, there were 1,50,000 deities installed in 2018 and this number is increasing year by year. This consistent increase in number for Ganesha's makes this problem a significant problem to be addressed. This is being addressed in various means by increasing the timings of local trains in the procession days etc, but due to various other factors like consistent growth in urbanization, the issue still persists. During the procession of Ganesha idols to various immersion points located at various locations throughout the city and all over the country, cause traffic problems as mentioned later. On the eleventh day, the Ganesha idols is taken through the streets of a city in a procession accompanied by people singing and dancing and during this procession, traffic is very difficult to manage which is results into traffic congestion. Congestion here refers to many factors inclusively like the extra delay caused over the time taken by common people to reach their destination, the traffic disruption caused by people who come over the roads to witness the eve of Ganesha immersion. The specific questions to be addressed include should they change their path to reach their destinations or should the Ganesha idols change his way so that common people's routine should not go disturbed and also the idols occupying the roads in an scattered manner is minimized, etc. In simple terminology the problem here can be stated as finding the balance between the count of Ganesha idols to be immersed and specific lake, and optimizing the path of Ganesha idol's procession to the lake which must be done in a very effective manner and to ensure that the day to day routine of common people should not be disturbed. In Section II we have discussed about the related works that had been done. Section III consists of Methodology that we have used to address and solve the discussed problem. In this apart from giving one proper solution, we have compared different strategies we have tried. In Section IV we have done Results and Analysis of solutions we discussed different in Section III. And in Section V we have concluded our finding and future work together as Conclusion.

II. RELATED WORK

To the best of the authors knowledge, this specific problem has not been addressed in the literature. In this section, few related works on congestion control of traffic and Routing/Re-Routing problems is discussed.

One of such paper titled ***An approach to avoid traffic congestion*** Their approach aims to find the optimal path any time, avoiding queues and congestion, and helps improve traffic flow and road safety. This approach is divided into two main parts that can be run in parallel, the first part is the data storage and the second is the analysis and calculation of these data. The first part is to collect and record the movement of

all vehicles in the road, they only record Ids of roads and the date and the change for each vehicle. The second part is performed in two phases:

Phase 1 is to create a graph showing the map where the edges match to roads, peaks at the intersections of roads, and the weights of the edges in time to travel. In addition to the map of the other necessary information such as information about the traffic, the maximum speed in each segment and the speed of the vehicle in question and the data recorded in the first part.
Phase 2 is to apply Dijkstras algorithm to generate a graph to determine the optimal road from the current vehicle position to the destination point.
It cannot be directly applied to the problem statement given so we have formulated some other approach.

III. METHODOLOGY

The problem discussed before need to be addressed. We have designed an algorithm that uses the concept of shortest path and MST that works completely fine for the solution.

A. Solution Prerequisite and Overview

This paper gives various possibilities to solve this problem using classification concepts like clustering along with various graph theory concepts and graph related algorithms. Here, we are taking input from maps and transforming information from the maps into graphs so as to apply graph theory concepts. Various graph related concepts discussed here are Dijkstra's algorithm, minimum spanning tree (MST) and various results obtained through empirical calculations. The proposed Solution consists of 3 phases:
→ **Clustering**
→ **Balancing**
→ **Finding and Optimizing the path.**

B. Clustering

Clustering here referred as a clustering of Ganesha idols to it's optimally nearer lake. As in proposed algorithm, we need to classify various Ganesha idol into various sets, each set defines the set of Ganesha idols that are routed to specified immersion point. The Ganesha idols are classified based on the distance between the Ganesha idol and immersion point i.e. lake. The input is taken from Google maps using the Google maps API, the input format is taken as the distance between the Ganesha idol and all the immersion points. Now based on this distance, each Ganesha idol is classified to each lake. In this manner, clustering is done and each cluster consists of Ganesha idol and each cluster of Ganesha idols is routed towards a specific lake point.
Dijkstra's algorithm is used to calculate when no input is obtained from Google maps API or if we have a manual map. In such cases, we have to plot the junctions (chowpattis), Ganesha idols and immersion points. The next task is to we have to compute the distance between each Ganesha idol and Lake using Dijkstra's algorithm and now using this distance as base distance we have to cluster the Ganesha idol into possible cluster. After clustering, there is a possibility that most of the Ganesha idols are transferred to only few lakes and remaining lakes are left idle. In such cases, balancing need to be done considering various other factors and various empirical results are used to balance the Ganesha idols.
Algorithm1 gives details about the clustering phase.

Algorithm1 Pseudo Code for ***Clustering*** phase

```
while i in number of Ganeshas satisfy do
   initialize min and lakeId
   for each lake bf in number of Lakes do
      if(min > ganeshalakedistance[j][i].distance)
         min = ganeshalakedistance[j][i].distance;
         minLakeId = ganeshalakedistance[j][i].lake;
         end if
   end for
      ganesha[i].laketoid = minLakeId;
end while
```

In *Algorithm 1*, ganeshalakedistance is a two dimensional matrix which has the distance between every Ganesha idol and immersion point.

Balancing here refers to rectifying the imbalances that were created in the process of clustering. In the process of clustering the Ganesha idol into various immersion points there are several optimal clustering. Among them the best possibility is when the Ganesha idol's are clustered in such a way that each immersion point gets equal number of Ganesha idols, but that possibility may not occur always. Balancing can be defined as a process of rectifying the imbalances in such a way that the new set of cluster should be near to the best possibility.
The basis of balancing is the difference between the number of Ganesha idol i.e; every immersion point will have certain number of Ganesha idol to be immersed, and the difference among them is calculated. Now, based on this difference the balancing is done. Based on empirical calculations we obtain three conditions, on which algorithm is implemented and verified which condition suits the most. Those three conditions are

$\rightarrow maxCount > (2 * minCount)$
$\rightarrow (2 * maxCount) > (3 * minCount)$
$\rightarrow (2 * tempMinCount) < maxCount$

where, maxCount refers to the number of Ganesha idols are highest at an immersion point which may or may not change after the balancing, minCount refers to the number of Ganesha idols are minimum and greater than zero and tempMinCount refers to the temporary variable which holds

the minimum distance value that has been swapped from cluster with more number of Ganesha idols to the other cluster of Ganesha's. In either of the conditions, Ganesha idols are swapped from cluster with larger number of Ganesha idols to another cluster. The swapping operation follows a certain pattern which is discussed in further sections of the paper. The pattern in general is to arrange all the immersion points in ascending order of distance to each lake and shift one Ganesha idol from the immersion point i.e. lake with maximum count of Ganesha idol's to the immersion point i.e. lake with count just less than max count and the same goes on further till it reaches immersion point with minimum number of Ganesha idols. This whole process of Shifting is done till it violates the conditions mentioned above. Once the condition is violated, the maxCount and minCount is computed again and checked iteratively till the Ganesha's are balanced.
Now we look into each condition in detail.

i) $maxCount > (2 * minCount)$:
In this particular condition, both the variables changes through out the balancing process i.e: maxCount and minCount changes continuously for every iteration till the process ends. The balancing process starts when the maximum number of Ganesha idols is twice the number of Ganesha idols for any immersion point. Once this condition is satisfied, then the Ganesha idol should shift from the immersion point to the previous one in sequence that is arranged after checking the condition.
ii) $(2 * maxCount) > (3 * minCount)$
In this condition both the variables changes continuously through out the balancing. Similar to the above condition, the balancing process starts once the condition is satisfied and continues till the condition is violated. Once the condition is violated, then both the variables are computed again and checked for the mentioned condition. In this way, we check for the condition for all the immersion points in an iterative process.
$(iii)(2 * tempMinCount) < maxCount$
In this particular condition, unlike afore the above mentioned conditions, one of the variable is static while the other variable is updated continuously. tempMinCount refers to the temporary variable which holds the minimum distance value that has been swapped from cluster with more number of Ganeshas to the other cluster of Ganesha and maxCount refers to the number of Ganeshas are highest at an immersion point which may or may not change after the balancing. maxCount is fixed through out the process and its value is computed again, unlike to the previous variables where their respective values are computed for every inner loop iteration.

On the basis of experiments conducted, it was observed that the first condition ($maxCount) > 2 * minCount$) derives the best solutions. Once the balancing is done the next step is to find the minimum path covering all the Ganesha idols to the lake with minimum possible congestion. To obtain such a way, we used the concept of minimum spanning tree.

The next step is to select the Ganesha idol to be immersed. There are two approaches to do this:
a. Shift the Ganesha idol at a Distance to lakes of mincount and to that of lakes of maxcount is minimize.
b. Shift the Ganesha idol, that is presently not to lake of mincount, at a least Distance to lakes of mincount, i.e. the closest Ganesha idol is swapped to mincount Lake.

We have discussed these in detail.

C. Minimum Spanning Tree

Minimum Spanning Tree is defined as the subset of a Graph, which has covered all the edges with minimum possible edges and minimum possible weights. In the current scenario, once the clusters are formed and balanced, then among clusters a minimum spanning tree is constructed and in this tree, the Ganesha idol at minimum distance from immersion point is found and it is connected to the immersion point . To find the minimum spanning tree, we took the distance from Google maps API and used them to find MST and joined the Ganesha idol, at least distance from immersion point, to the immersion point.

IV. RESULTS AND ANALYSIS

In the implementation JavaScript was used to build front end and Java Servlets for back end coding of the web application to test the algorithm. And Google map API to work on real map.

Consider the following diagrams:

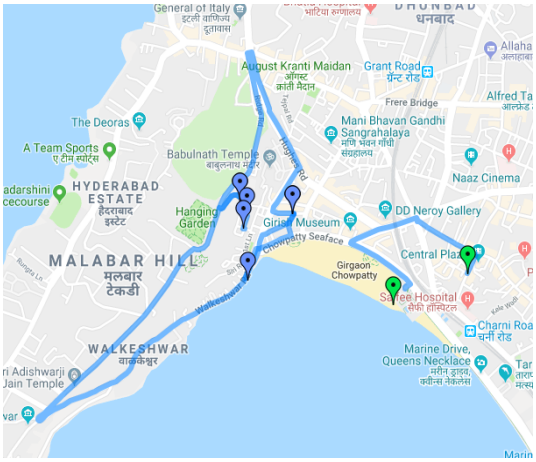

Fig 1. Without applying Opt MST and no balancing

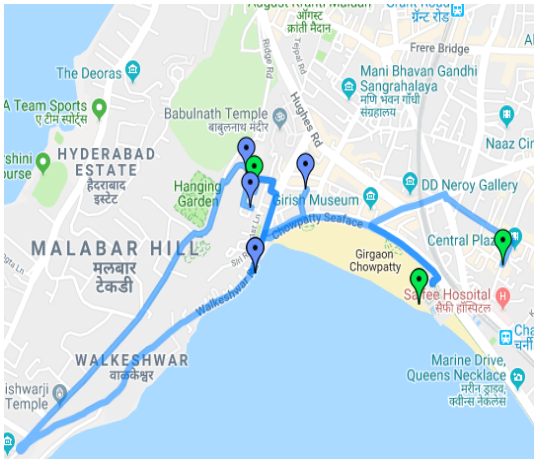

Fig 2. No Opt MST with balancing Approach 1

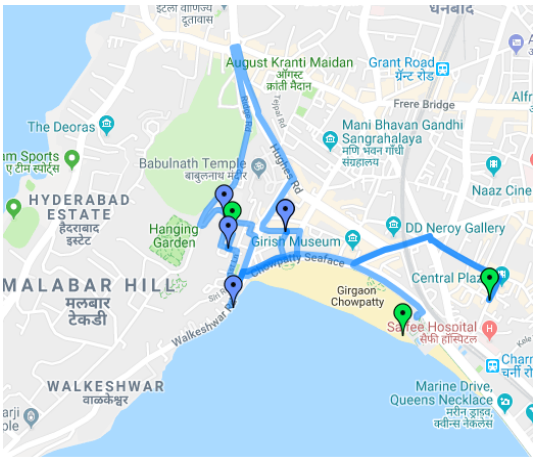

Fig 3. Applying Opt MST with balancing Approach 1.

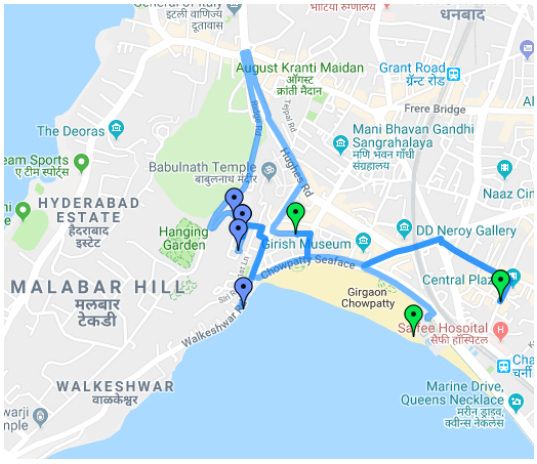

Fig 4. Applying Opt MST with balancing Approach 2.

As we have shown in proposed algorithm, we have discussed solution as:

1. Balancing on the count of number of Ganesha idols to be immersion in one lake (immersion point).
There are two approaches to do this:

a. Shift the Ganesha idol at a Distance to lakes of mincount and to that of lakes of maxcount is minimize.
b. Shift the Ganesha idol at a Distance to lakes of mincount i.e. the closest Ganesha idol is swapped to mincount Lake.

2. Optimised MST Algorithm is used for finding optimal route from Ganesha idols to lake (choosen).

Let ***optdist*** be the optimised distance that we found by applying Approach 1.a. and Approach 2 as stated above.
Different combinations can be derived from above mentioned possibilities.

Case 1. Without applying MST and no balancing
In this proposed combination of approaches presents the results obtained by using the distance measures obtained directly from the Google Maps API, without using the proposed approach. As shown in Fig 1, it was found that the distance or path traversed is very high. i.e. approx. twice of **optdist**.

Case 2. No MST with balancing Approach 1 i.e 1.a
In this combination of possibility, we are only doing balancing and let Google Maps API get us the path between the Ganesha idols and the lake. The paths we found here were to 2 times of the **optdist**. This is shown in Fig 2. Hence, we needed more improvement. This shows balancing is not enough for reducing distance and road engaged in this procession. So, we tried case 3.

Case 3. Applying MST with balancing Approach 1 i.e 1.a as stated above.
We are applying Optimized MST Approach along with the balancing Approach 1. The results were surprisingly better. The results were around 1.2 to 1.5 times of the optdist.This indicates that the intuition of applying MST is justified. So, we needed now optimized balancing Approach. Approach 1.a. is having a serious overlapping of path issues as we can see in Fig 2 and Figure 3. So we tried Approach 2 for balancing.

Case 4. Applying MST with balancing Approach 2 i.e 1.b as stated above.
We are applying Optimized MST logic along with the balancing Approach 1. The results were surprisingly better that past approaches. The results were best so far and we called it, **optdist.** This is showing our approach of Approach 2 did work better than Approach 1. The best result is obtained using this approach. This is shown in Figure 4.

Need to emphasize here is, Approach 1 i.e. 1.a and Approach 2 i.e. 1.b for balancing both works exceptionally well in different cases of given map but, most of the cases Approach 2 i.e. 1.b gives better results.

V. CONCLUSION

In this paper, we have discussed applications of various computer science concepts to solve real time problems. In this paper, the real time problem was reduction of congestion during Ganesha idol Immersion.

The approaches we took to solve the problem is explained in this paper. We have discussed all the approaches in detail, with comparative resulted path on the same location of Ganesha idol and lake, so that we can compare them on same parameters, although we have tested for many possible maps. The results were very promising with the *balancing logic 1.b. and Opt MST* used together. We have optimized the path to half the Google maps direct paths.

There are many similar procession all over the country. This approach can be applied with any particular specification immaterial of the location with the necessary condition that rate you can decide the edge weight.

REFERENCES

[1] Hamza TOULNI, Benayad NSIRI, et. al., Mohammed BOULMALF, Mohammed BAKHOUYA, et. al.
An approach to avoid traffic congestion:
2014 Fifth International Conference on Next Generation Networks and Services (NGNS) May 28-30, 2014, Casablanca, Morocco

[2] Dijkstras algorithm,
`https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/`
as Retrieved on: 17/11/2018

[3] Prims Minimum Spanning Tree (MST) — Greedy Algo,
`https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/`
as Retrieved on: 17/11/2018

[4] Google Maps - Google Cloud Platform
`https://console.cloud.google.com/projectselector/google/maps-apis/`
as Retrieved on: 17/11/2018

[5] Google Maps Basic,
`https://www.w3schools.com/graphics/google_maps_basic.asp`
as Retrieved on: 17/11/2018

[6] Printing Paths in Dijkstras Shortest Path Algorithm,
`https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm`
as Retrieved on: 17/11/2018