Shape Based Sub Trajectory Similarity Matching

Pranali Yawalkar
CS11B046
Indian Institute of Technology
pranali.yawalkar@gmail.com

Rishitha Dega
CS11B035
Indian Institute of Technology
rishitha.reddy.3@gmail.com

ABSTRACT

Time Series data are used in statistics, signal processing, pattern recognition, econometrics, weather forecasting, etc and largely in any domain of applied science and engineering which involves temporal measurements. Spatial Time Series data are used in analysis where observations typically relate to geographical locations. E.g. obtaining the trajectory of users and cars using the GPS sensors. For analysis we define a data model of trajectories as directed lines in a space, and the similarity between trajectories is defined based on the shape of the trajectories in the spatiotemporal data model. The analysis model proposed in this project focuses on building a robust structure to match a query trajectory with sub trajectories of the trajectories in the database. Since querying the huge database is computationally expensive, we are coming up with a new indexing model to efficiently retrieve trajectories whose shape is similar to the shape of a candidate trajectory from the database.

Keywords

Subtrajectory Matching, Time Series data, Clustering

1. INTRODUCTION

Trajectory analysis is a topic of growing interest with many real world applications. Finding similar trajectories in spatial time series is accompanied by its high cost of computation. Finding matching sub trajectories becomes all the more a difficult task and non scalable. Often exact match for finding similarity is not strictly necessary and this project tries to leverage this constraint of exact match. The project discusses about finding shape wise similar sub trajectories with efficient indexing. Consider some of the applications like estimating Traffic Jams (Refer to Figure 1) : Here distance metric is used to find close-by trajectories and shape metric to check if the trajectories overlap on the same road. Our model can be extended to use the distance closeness as well. Sub-trajectory matching will be useful to identify the time duration where the trajectories overlapped.

Consider another example of weather forecasting (Refer to Figure 2). Given the weather vs time data for so many years, and given a time-series query observed in the present, we can use the query to match the top-k similar shapes in the database to predict the weather. Here, full trajectory matching is done instead of sub-trajectory matching. Our model works on spatio-temporal data and can be reduced to work on time-series data.

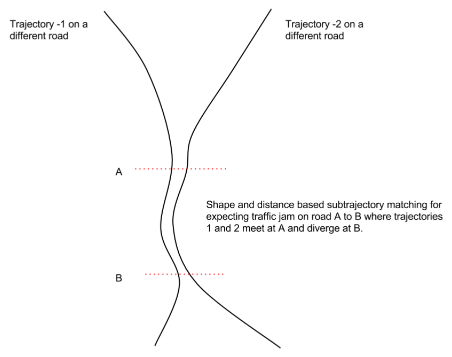
Trajectory -1 on a different road
Trajectory -2 on a different road
A
B
Shape and distance based subtrajectory matching for expecting traffic jam on road A to B where trajectories 1 and 2 meet at A and diverge at B.

Figure 1: Expeting Traffic jams

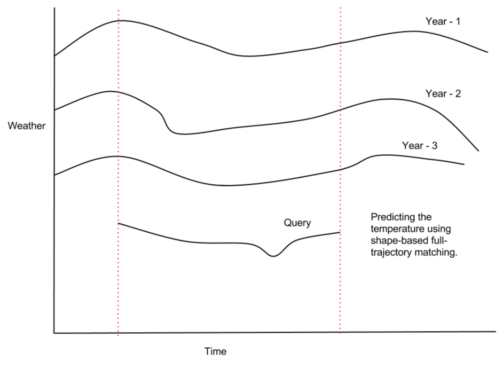

Figure 2: Weather forecasting

Also consider one of the the hottest topics of Trend Analysis . Given the time-series data of an aspect, say sensex points for so many companies, or following trends in twitter, and given a query pattern, we can use shape-based full-trajectory matching to find if the pattern was trending in that time duration based on some thresholds for trends similarity.

2. RELATED WORK

Shape-based Similarity Query for Trajectory of Mobile Objects [1]: uses L2 distance for finding the shape based similarity. It focusses on finding all the subsequences of the database trajectories of length equal to query's length that match the best with the query. It uses R^+ trees for indexing the data it works on range queries to find the most similar trajectories to given query trajectory.

Indexing Spatiotemporal Trajectories with Chebyshev Polynomials [4] gives novel indexing techniques for subsequence matching. It deals with range queries (can be extended to KNN) and talks about dimensionality reduction techniques and MBR (minimum bounding rectange) indexing technique.

Similarity Search Algorithm for Efficient Sub-trajectory Matching in Moving Databases[8] paper talks about a variant of the dynamic distance warping algorithm to reduce the run time of exisiting warping distance algorithm. It deals with shape based subtrajectory matching using a matching function based on the maximum and minimum distance between the two trajectories under consideration. It uses dynamic programming with backtracking to find the matching subtrajectories.

The motivation of the problem is to come up with a novel scalable technique for finding subtrajectory matches based on shape similarity. There hasn't been much work in the field of finding subtrajectories on a non metric distance function. The baseline technique faces problems with complexity and doesn't scale well. The project proposes a heuristic technique which trades off accuracy for runtime development and tackles the issue of scalability with minimal accuracy loss. The solution is evaluated well for its accuracy and runtime and a novel indexing technique is proposed.

3. PRELIMINARIES

In this section, we introduce the concepts that are central to the techniques developed in the paper.

Definition 1: A trajectory T = $\{(s_0, t_0), (s_1, t_1), (s_2, t_2)$ ... $(s_m, t_m)\}$ is a set of points where each $s_i = x_i, y_i$ is a point in 2D space at time t_i. Refer to figure 3.

Definition 2: Piece wise approximation. A technique used to find the missing points by interpolating the data. Refer to figure 4.

Definition 3: Temporal Normalized Discrete Trajectory. The trajectory T_{new} obtained by applying piecewise appoximation to a trajectory T. Refer to figure 5.

Definition 4: Simlilarity Metric. A numerical quantity to estimate how shape-similar two trajectories are in the spatio temporal representation.

3.1 Piecewise approximation

Let S$_T$ be the collection of all the time for which T has the GPS data. Eg, if T = {(s$_0$, t$_0$), (s$_1$, t$_1$), (s$_2$, t$_2$), (s$_4$, t$_4$)}, then S$_T$ = {0, 1, 2, 4}. T(t_i) = s$_i$. Here we can see that data at time 2 is missing. This is because the data captured using GPS is not available at crisp regular intervals.
Therefore we use Piece wise approximation technique to interpolate the data and sample at regular intervals to get the temporal normalized discrete trajectory. We are sampling the data at intervals of 5 minutes. Thus the new temporal normalized discrete trajectory T$_{new}$ is :

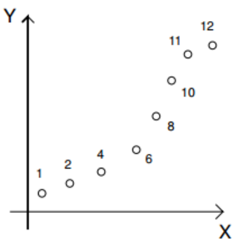

Figure 3: Original data with missing values

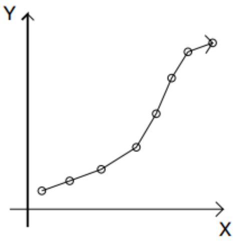

Figure 4: Approximated data

$$T_{new}(t) = \begin{cases} T(t) : t \in S_T \\ \dfrac{t - t_i}{t_{i+1} - t_i} T(t_i) + \dfrac{t_{i+1} - t}{t_{i+1} - t_i} T(t_{i+1}) \quad : \\ t \notin T_W \text{ and } t_i < t < t_{i+1} \end{cases}$$

In the preprocessing step we apply piecewise approximation to all the trajectories in the database and use the obtained temporal normalized trajectories for later computations. Similarly query trajectory is also transformed into temporal normalized discrete trajectory.
All the subtrajectories of the database trajectories overlapping temporally with the query T$_Q$ are retrieved and the shape similarity is calculated.

In Figure 6 as we can see, only trajectories T$_1$ and T$_2$ overlap with the entire query length in time and hence the sub-trajectories of T$_1$ and T$_2$ lying between the two read vertical lines are considered and matched against the query trajectory for similarity.

Figure 4 shows the original data from GPS having missing values. Figure 5 shows an approximation technique. Figure 6 shows the temporal normalized trajectory.

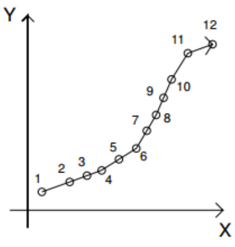

Figure 5: Temporal Normalized Trajectory

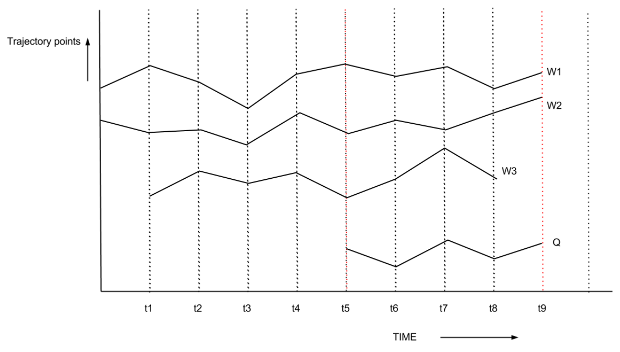

Figure 6: Similarity description

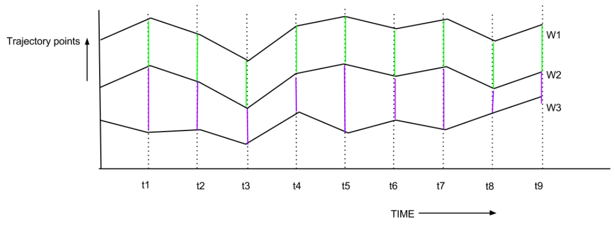

Figure 7: Variance description

4. SIMILARITY METRIC FORMULATION

The similarity metric used to find shape based similarity between two trajectories T$_1$, T$_2$ of same size between time interval t$_i$ to t$_{i+m}$ is based on how varying the L2 distances between the corresponding points of the trajectories are i.e the variance in the distances between the pairs (T$_1$ (t$_i$), T$_2$ (t$_i$)), (T$_1$ (t$_{i+1}$), T$_2$ (t$_{i+1}$)), (T$_1$ (t$_{i+2}$), T$_2$ (t$_{i+2}$))..., (T$_1$ (t$_{i+m}$), T$_2$ (t$_{i+m}$)). Similarity is being captured by the fact that if two trajectories are having exactly the same shape, then their points at a given time will be as far apart as points at any other time. Less variance among these distances show that the points in the two trajectories are having less variation in the distances between them at different times and hence more similar shapewise.

Refer to figure 7, we can see that T$_2$ and T$_1$ are more similar and the variance in the distances between their points at different times is less. T$_2$ and T$_3$ are less similar shapewise because the variances in the distances between their points are different times is more.

$$D(T_1, T_2, t_1) = L_2((T_1(t_1), (T_2(t_1))$$

$$Sim(T_1, T_2) = \{D(T_1, T_2, t_1), D(T_1, T_2, t_2), ...D(T_1, T_2, t_n)\}$$

$$Variance(T_1, T_2) = \sum_{i=1}^{n}(D(T_1, T_2, t_i) - Mean)^2/n$$

$$Similarity(T_1, T_2) \propto 1/Variance(T_1, T_2)$$

Algorithm 1 Similarity between trajectories
Input: Trajectory T$_1$ = $\{(s_{11}, t_{11}), (s_{12}, t_{12}) ...(s_{1m}, t_{1m})\}$ and Trajectory T$_2$ = $\{(s_{2p}, t_{2p}), (s_{2p+1}, t_{2p+1}) ... (s_{2p+l}, t_{2p+l})\}$.
Output: Similarity between T$_1$ and T$_2$

```
1: $t_{start} \leftarrow t_{2p}$
2: $t_{end} \leftarrow t_{2l}$
3: flag ← 0
4: for each (s$_{1x}$, t$_{1x}$) in T$_1$ do
5:     if t$_{1x}$ = t$_{start}$ then flag = 1;
6:     end if
7: end for
8: if flag & $t_{2l} < t_{1m}$ then
9:     similarity ← List of size = l
10:    Mean ← 0.0
11:    Var ← 0.0
12:    for 0 ≤ j < l do
13:        s$_{match}$ ← $s_{1x}$ where $t_{1x} = t_{2p+j}$
14:        temp ← distance(s$_{2p+j}$, s$_{match}$)
15:        Mean += temp;
16:        similarity.add(temp)
17:    end for
18:    Mean = Mean/l
19:    Var = $\sum_{j=0}^{l}(similarity.get(j) - Mean)^2/m$
20:    return 1/(Var+1);
21: end if
22: return 0
```

5. PROBLEM FORMALIZATION

In this section, we formally define the problem.

Definition 1: Trajectory Matching. Given a query trajectory T$_q$ and the datebase of N trajectories D = {T$_1$, T$_2$, T$_3$,..T$_N$}, the problem is to find the trajectory T$_i$ which is most similar to T$_q$. The trajectory that minimises its variance with T$_q$ using the above defined similarity metric [**Similarity(T$_i$, T$_q$)**] is the match.

Definition 2: Sub Trajectory Matching. Given a query trajectory T$_q$ and the datebase of N trajectories D = {T$_1$, T$_2$, T$_3$,..T$_N$}, the problem is to find the subtrajectory T$_i$ of T$_q$ which has the highest similarity to subtrajectory T$_j$ of trajectory T$_k$ in D . The subtrajectory T$_i$ that minimises its variance with subtrajectory T$_j$ of trajectory T$_k$ using the above defined similarity metric [**Similarity(T$_i$, T$_j$)**] is the match.

6. BASELINE TECHNIQUE

Naive way to find the subtrajectories in the query to match against database trajectories is to iteratively consider all possible subtrajectories of all possible time duration and matrch against the corresponding subtrajectories (in the same time duration) of all trajectories in the database.
Complexity analysis :

- All possible subtrajectories : $\mathcal{O}(n^2/g^2)$ where n is the number of points in the query trajectory
- Number of trajectories in Database : $\mathcal{O}(N)$
- Similarity computation between a possible subtrajectory and query subtrajectory = $\mathcal{O}(n)$

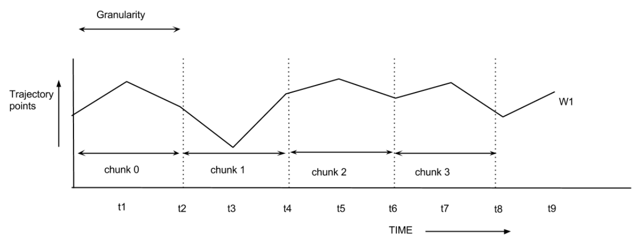

Figure 8: Similarity description

- Total number of computations of similarity : $\mathcal{O}(n^3N/g^2)$

Algorithm 2 Sub-Trajectory Matching (Baseline)
Input: Query trajectory T$_q$ $\{(s_{q1},t_{q1}),\ (s_{q2},t_{q2})\ ...\ (s_{qm},t_{qm})\}$ and G (Granularity).
Output: Top-K matches of query trajectory

```
 1: Similarities ← Array of Maps
 2: for each trajectory T$_i$ $\{(s_{i0},t_{i0}),(s_{i1},t_{i1})...(s_{in},t_{in})\}$ in Database do
 3:     j ← 0
 4:     for each l = j*G (multiples of G) do
 5:         Similarity[j].get(T$_i$) ← MIN_VALUE
 6:         for each sub-trajectory T$_s$ in query of size l do
 7:             sim = Similarity(T$_i$, T$_s$) - Algorithm 1
 8:             if sim > Similarities[j].get(T$_i$) then
 9:                 Similarities[j].get(T$_i$) ← sim
10:             end if
11:         end for
12:     end for
13: end for
```

7. INDEXING BASED SOLUTION

Lets discuss about two terms which quantify the trade off between accuracy and runtime. Indexing structure is proposed to enhance the **Pruning power using k-means clustering** which would save us unnecessary computataions greatly bringing down the run time. **Granularity** talks about how much bias are we having over the exact solution. There is no direct relation of granularity with the accuracy, but a smaller granule is more likely to perform better. These two quantities **l** (number of clusters of the k-means clustering) and **granularity** are learnt using cross validation set. A heuristic solution to the highly complex problem is obtained by relaxing on the accuracy based on an assumption of local maximas. Consider a query trajectory T$_q$ = $\{(s_0,t_0),(s_1,t_1),(s_2,t_2)\ ...\ (s_m,t_m)\}$ obtained after the piecewise approximation interpolation. Instead of considering the whole trajectory at once, we are dividing the trajectory into so many chunks of size of granularity. The best value of granularity which will define the number of chunks will be decided using cross validation. The essence of granularity is that it defines the minimum size of the final subtrajectory match that is produced. A subtrajectory match, as per the proposed technique, will never be of a shorter length than the granularity specified. Refer to Figure 8 for the break down of the query into chunks using chunk granularity. We then indepedently identify top matches for each chunk iteratively. Pruning away trajectories from D which are very dissimilar to the query chunk helps reduce the query time majorly. This is done by using **k-means clustering** as a part of offline preprocessing.

7.1 Offline Indexing

- All the trajectories are divided into chunks.
- For each chunk the portion of all trajectories lying in that chunk are clustered based on the shape similarity.

7.1.1 Complexity Analysis

Consider granularity = g and number of clusters = l which are iterated over in the experiment to find the best pair using cross validation

- Total time spanned by the trajectories in D : T$_{span}$
- Total number of trajectories in D : $\mathcal{O}(N)$
- Total number of chunks in D : $\mathcal{O}(T_{span}/g)$
- Maximum total number of elements in each chunk : $\mathcal{O}(N)$
- Similarity computation between two queries of size g : $\mathcal{O}(g)$
- Clustering complexity for each chunk : $\mathcal{O}(NglI)$ where I is the number of iterations of the k-means convergence.
- Total complexity : $\mathcal{O}(NT_{span}lI)$

Algorithm 3 Offline Indexing
Input: N(Number of clusters), G (Granularity) and Database of Trajectories D
Output: Clusters of sub-trajectories C

```
 1: M ← #Chunks
 2: for each T$_i$ = $\{(s_{i1},t_{i1}),\ (s_{i2},t_{i2})\ ...(s_{im},t_{im})\}$ in D do
 3:     S$_{i1}$, S$_{i2}$, ... S$_{iM}$ ← Divide T$_i$ into sub-trajectories each of size G.
 4: end for
 5: for 0 ≤ j < M do
 6:     Assign random centroids to N clusters
 7:     do
 8:         Assign each subtrajectory in chunk j the closest cluster i.e maximum Similarity(S$_i$, C$_i$).
 9:         Compute new centroids of N clusters.
10:         Repeat till the centroids converge.
11:     while Centroids do not converge
12:     C$_j$ ← N clusters
13: end for
14: C = {C$_1$,C$_2$, ... C$_M$}
```

7.2 Online Processing

Online Processing :

- On receiving the query and breaking it down into chunks, each query chunk is examined to find the cluster it belongs to
- The elements in this cluster would be more shape similar to this query chunk than the rest of the elements in other clusters saving us the computation of similarity with so many other elements. Similarity is computed between query chunk and all the other elements that lie in that cluster
- The top-k (k is chosen as 50 in the experiment) are computed. After doing this iteratively for each query chunk, we now have top-k database trajectories' chunks for each query chunk which are most similar to it

- We now consider top-3 (among the top-k) trajectories from each chunk iteratively and find the longest contiguous sequence of chunks which have the trajectory under consideration in its top-k

Experimental evaluation section covers the values pertaining to reduction in computation owing to clustering.

7.2.1 Complexity Analysis

- Worst case size (in terms of number of elements) of a cluster across all chunks = N. This will happen when all the elements in a chunk belong to one cluster.
- Similarity computation between two queries of size g : $\mathcal{O}(g)$
- Number of query points = n
- Number of query chunks = n / g
- Finding the cluster belongingness of a query chunk : $\mathcal{O}(l * g)$
- Finding the cluster belongingness of all query chunks : $\mathcal{O}(l * n)$
- Number of computations between each query chunk and the elements in its cluster : $\mathcal{O}(Ng)$
- Number of compuataions for all such query chunks = $\mathcal{O}(nN)$
- Total : $\mathcal{O}(ln)$ + $\mathcal{O}(nN)$

Algorithm 4 Sub-Trajectory Matching
Input: Query trajectory T$_q$ $\{(s_{q1}, t_{q1}), (s_{q2}, t_{q2}) \dots (s_{qm}, t_{qm})\}$
Output: Top-K matches of query trajectory

```
1: Chunks ← Clusters of sub-trajectories Offline Indexing
2: Divide the query into sub-trajectories S each of size G.
3: for each S_i $\{(s_{ip}, t_{ip}), (s_{ip+1}, t_{ip+1}) \dots (s_{ip+G}, t_{ip+G})\}$ do
4:     C_i ← Chunks[i]
5:     c ← closest cluster in C_i to S_i.
6:     (i.e centroid of cluster c has maximum similarity with S_i)
7:     Find k trajectories T_j in cluster c which have top-k maximum Similarity(T_j, S_i)
8: end for
```

8. EXPERIMENTAL EVALUATION

8.1 Evaluation Technique

To test the correctness of the approach, we tested it on two sets each of 30 trajectories - Test1 data and Test2 data. The process followed :

- The best match is found using Baseline technique = match$_1$
- The best match is found using the proposed approach = match$_2$
- If the similarities that these two subtrajectories have with the query's subtrajectories are comparable, we are classifying it as a good match
- Comparable metric : Say the variance of match$_1$ is V$_1$ and that of match$_2$ is V$_2$. If $\frac{max(match_1, match_2)}{min(match_1, match_2)} < \epsilon$, then we say that the variances are comparable and match$_2$ is a good enough substitute for match$_1$
- Accuracy of the algorithm on the test data is obtained as a percent of suitable matches found (as per the comparable metric) over all matches found.

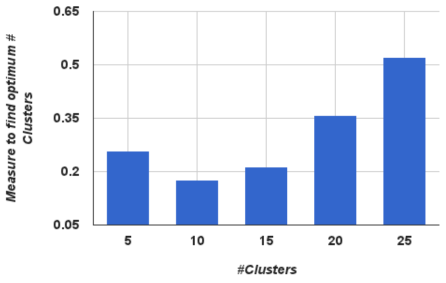

Figure 9: Finding Optimal #Clusters

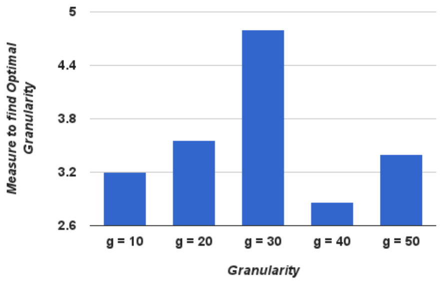

Figure 10: Finiding Optimal Granularity

8.2 Cross Validation

To capture the optimal parameters of l, g and ϵ we have used cross validation on a data of size 300 trajectories. The indexed data is of size 4000 trajectories.

- Optimal l : the one giving lesser runtime, lesser variance of the answer and longer subtrajectory answer. Essentially it is the one that minimizes (runtime*variance / length). Based on the experiments optimal l is chosen as 10. Refer to Figure 9.
- Optimal g : g value affects the accuracy of the algorithm . It also affects the runtime for the baseline technique. As stated before, g poses a constraint on the minimum size that the final subtrajectory match can have. It imposes that the size of the final answer be of multiples of g and should start at appropriate g-divided chunk markings. Based on the experiments optimal g is chosen as 40. Refer to Figure 10. There is no extablished relation between g and accuracy. It is a property of the data and is subject to vary with it. g value is calculated by minimising (runtime*variance / length) for cross validation set.
- Optimal ϵ : match$_1$ and match$_2$ are computed for every query in the cross validation set. Value $\frac{max(match_1, match_2)}{min(match_1, match_2)}$ over all queries is computed. The optimal ϵ is chosen graphically from a trade off between minimising the number of outliers and minimum epsilon value. Figure 11 shows that optimal value of ϵ is chosen as 50. The value of g = 40 and k = 15 is chosen.

8.3 Index Construction

The plots for index construction time for l and g values are shown in Figure 12 :

8.4 Online Query Performance - time

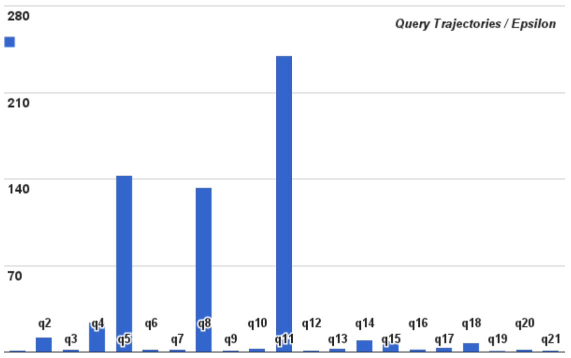

Figure 11: Epsilon for trajectories (Cross Validation)

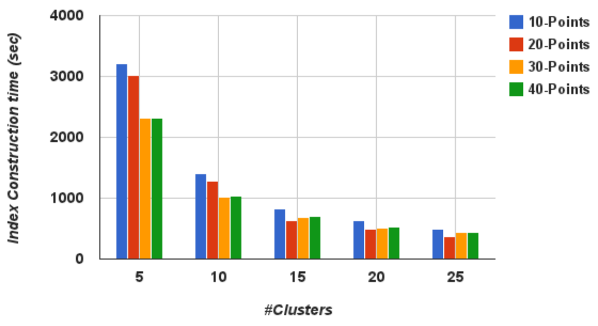

Figure 12: Index construction time vs #Clusters for different g

The plot of query time vs l for various in Figure 13. We can see that query time cleanly decreases with increasing k because of more pruning.

8.5 Online Query Performance - Comparison

The table for Percentage of comparable matches for the epsilon value when tested on Test1 data and Test2 data (g = 40 and epsilon = 50) :

	Test1	Test2
Baseline : Average runtime	53844.48	51106.6
Clustering : Average runtime	237.42	214.43
% Comparable matches :	66%	62%

8.6 Pruning quality

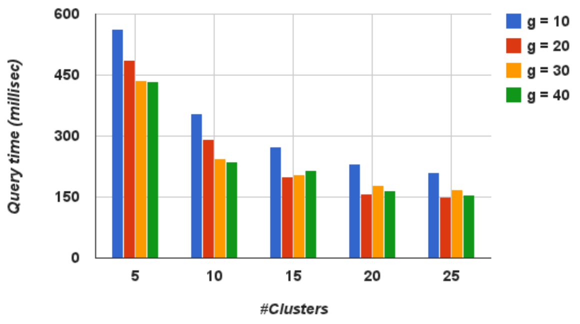

Figure 13: Query time vs #Clusters for different g

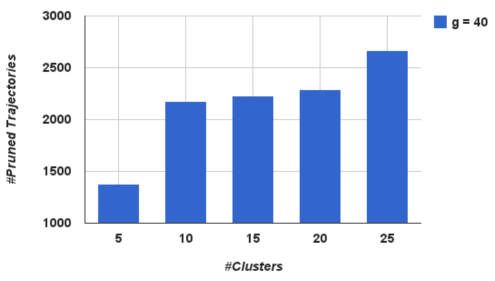

Figure 14: Pruned Trajectories vs #Clusters for test sets

The average number of elements pruned away for every chunk of the query for and varyign l and g for cross validation set, test1 data of longer queries and test2 data of shorter queries. Refer Figure 14.

9. CONCLUSIONS

From the experiment performed, we are able to propose a intutive and simple technique of tackling the subtrajectory matching problem on a non metric distance function. The algorithm greatly reduces the run time at the cost of minimal accuracy loss. The evaluation techniques discuss about this trade off in great detail. The theoretical complexity of the algorithm is way better than the baseline technique's. Considering that there hasn't been much work in the field of subtrajectory matching on a non metric distance function focussing on shape similarity, we would like to add some points which can be taken up as future modifications to this idea. When the data is high dimensional, we can extend the model by using DTW instead of L2 for finding the similarity. There is latent parallelisation in the problem. The chunk clustering is independently done and is one of the parallelizable parts.

10. REFERENCES

[1] Yutaka Yanagisawa, Jun-ichi Akahani, Tetsuji Satoh. "Shape-based Similarity Query for Trajectory of Mobile Objects"

[2] L.Chen, M.T. Ozsu, and V.Oria. "Robust and Fast Similarity Search for Moving Object Trajectories"

[3] Peng Chen, Junzhong Gu, Dehui Zhu and Fei Shao. "A Dynamic Time Warping based Algorithm for Trajectory Matching in LBS"

[4] Yuhan Cai. "Indexing Spatiotemporal Trajectories with Chebyshev Polynomials"

[6] Elias Frentzos, Kostas Gratsias, Yannis Theodoridis. "Index based most similar trajectory search"

[7] Xin Huang, Jun Luo, Xin Wang. "Finding frequent subtrajectories with time constraints"

[8] Eun-Cheon Lim, Choon-Bo Shim. "Similarity Search Algorithm for Efficient Sub-trajectory Matching in Moving Databases"