B.Sc. Engineering Thesis

Group Trip Planning Queries in Road Networks

By

Samiha Samrose

Student No.: 0805066

Mohammad Hafiz Uddin

Student No.: 0805004

Md. Iftekhar Mahmud

Student No.: 0805102

Submitted to
Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh

July, 2014

Declaration

Supervisor's Declaration

I confirm that, to the best of my knowledge, the research was carried out and the thesis was prepared under my direct supervision. It represents the original research work of the candidates. The contribution made to the research by me and by other members of staff of the University was consistent with normal supervisory practice. I have read this work and in my opinion this work is adequate in terms of scope and quality for the purpose of awarding the degree of Bachelor of Science in Computer Science and Engineering.

Tanzima Hashem, PhD
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000.

Candidates' Declaration

We confirm that, the work presented in this thesis entitled "**Group Trip Planning Queries in Road Networks**" is the outcome of the investigations carried out by us under the supervision of Assistant Professor Dr. Tanzima Hashem in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. We also declare that, neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

Samiha Samrose	Mohammad Hafiz Uddin	Iftekhar Mahmud
Student No.: 0805066	Student No.: 0805004	Student No.: 0805102

Contents

List of Figures

List of Tables

Acknowledgments

First of all, we would like to express our deepest gratitude to our supervisor Dr. Tanzima Hashem for introducing us to the fascinating world of spatial databases, guiding us through the path of conducting successful research and above all for always being there as our mentor. She shared her wisdom with us in analyzing subject matters and at the same time valued our thinking approach to synthesize those topics. Her suggestions drove us towards better ways of thinking, her reviews enriched us in solving research problems, and her support gave us strength at the time of our disappointment. We shall forever cherish the memories of working with her.

We would like to sincerely thank Sukarna Barua for his guidance and encouragement in developing our thesis work. Our very special thanks remain for Dr. Mohammed Eunus Ali for all his valuable reviews and remarks towards the dissertation. During the thesis work evaluation process, the teachers of our department made valuable comments which helped us a lot in refining our work. We express our gratitude to them all.

We are indebted to Md. Shahin Pramanik, a member of the Samsung Innovation Laboratory, who provided us with a congenial research environment in the lab while we were conducting our work there.

Last but not the least, we deeply thank our friends and families for always believing in us even at the moment when we were losing our believes in ourselves. We are forever grateful to them for their selfless support and understanding.

Abstract

The proliferation of location-based social networks allows people to access location-based services as a group. In this thesis, we introduce a novel approach to process Group Trip Planning (GTP) queries in road networks, which enable a group to plan a trip with a minimum aggregate trip distance. The trip starts from the source locations of the group members, goes via different point of interests (POIs) such as a restaurant, shopping center and movie theater, and ends at the destination locations of the group members. The aggregate trip distance can be the total or the maximum trip distance of the group members. We exploit elliptical properties to refine the search region for the POIs and develop efficient algorithms to evaluate GTP queries. Our experimental results show that our approach outperforms a naive approach significantly both in terms of query processing time and I/Os. Our approach is on average 17 and 40 times faster than the naive approach and requires 51 and 58 times less I/Os than the naive approach for minimizing total and maximum trip distances, respectively.

Chapter 1

Introduction

Location-based social networks such as Facebook and Google+ have enabled users to access location-based services (LBSs) as a group. An important class of group LBSs is a *group trip planning (GTP) query* that allows users to plan a group trip with the minimum travel distance. For example, a group of friends located at different places may want to meet at a restaurant, visit a shopping center, watch a movie together, and finally head to their homes. Specifically, a GTP query returns the point of interests (POIs) of required categories such as restaurant, shopping center that minimize the aggregate travel distance with respect to the source and destination locations of the group members. In this thesis, we propose the first approach to evaluate GTP queries in road networks.

GTP queries have been recently addressed in the Euclidean space [THK13]. However, the answer returned by an Euclidean GTP query algorithm does not always provide the minimum aggregate travel distance. In the Euclidean space, the distance between two points is measured as the length of the straight line connecting two points. In reality, user movement is restricted in a predefined road networks. Figure 1.1 shows an example scenario, where p_1 (movie theater) and p_2 (restaurant) minimize the total travel distance in a road network for a set of source-destination pairs $\{(s_1, d_1), (s_2, d_2), (s_3, d_3)\}$. The total travel distance is measured as the summation of each user's individual travel distance, where a user's trip starts at her source location, goes through the POIs and ends at her destination. A group may also want to minimize the maximum travel distance of the members. We focus on both of these aggregate functions SUM and MAX to evaluate GTP queries in road networks.

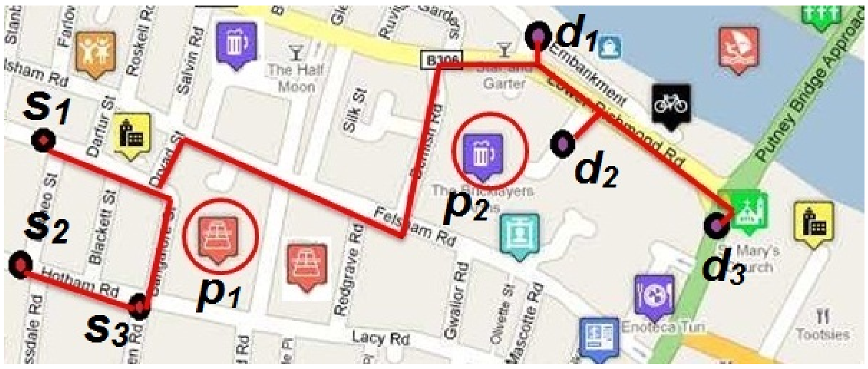


Figure 1.1: An example of a GTP query

1.1 Problem Formulation

A road network is typically represented by an undirected $G = (V, E, W)$ where V denotes the set of vertices (i.e., nodes), E denotes the set of edges. Each edge $(n_i, n_j) \in E$ is associated with a positive real number $w \in W$ that denotes the length of the road segment connecting the vertices n_i and n_j.

A road network also has a set of POI locations P where each POI $p_i \in P$ lies on an edge. POIs are categorized which means each POI p_i is associated with a category $C_i \in C$ where C is the set of all category sets available in a road network database.

Figure 1.2 shows an example of a road network that consists of six vertices, eight edges and weights assigned to each of the edges. The network shows two POIs p_1' and p_1'' on the edges (n_1, n_2) and (n_5, n_4), respectively. The edge between (n_1, n_2) weights 10.

A group trip planning (GTP) query is defined as follows:

Definition 1. k Group Trip Planning (kGTP) Queries: Given a road network $G(V, E, W)$, a set of n source locations $S = \{s_1, s_2, ..., s_n\}$, a set of n corresponding destination locations $D = \{d_1, d_2, ..., d_n\}$, sets of m categories of POI sets $C = \{C_1, C_2, ..., C_m\}$, k, and an aggregate function f, the kGTP query in road networks returns k sets of POIs $\{p_1^1, p_2^1, ..., p_m^1\}$, $\{p_1^2, p_2^2, ..., p_m^2\}$, $\ldots$, $\{p_1^k, p_2^k, ..., p_m^k\}$ that have k smallest values for f.

In a group trip of n users, the trip distance T_i of a user u_i is the total distance traveled by the user. This can be expressed by $T_i = d(s_i, p_1) + d(p_1, p_2) + \cdots +$

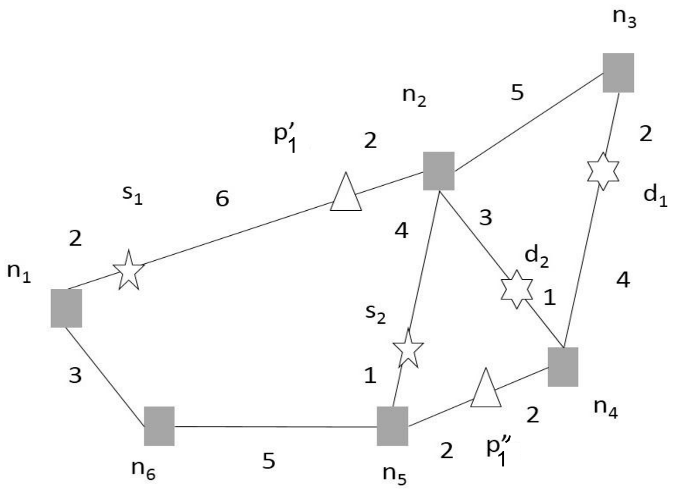


Figure 1.2: Road Network Representation

$d(p_m, d_i)$. We consider two variants of aggregate function f, SUM and MAX. If f represents SUM then the group trip distance GTD is determined as the summation of all T_is. If f represents MAX then the group trip distance GTD is determined as the maximum of all T_is.

Figure 1.2 represents a simplest form of a GTP query, where there are two users having source-destination pairs (s_1, d_1) and (s_2, d_2), and one asserted category $C_1 = \{p'_1, p''_1\}$. For SUM, a GTP query returns the point p''_1 from P because for p''_1 the minimum total travel distance value is 24. User-1 crosses distance value 18 along the path $s_1 - n_1 - n_6 - n_5 - p''_1 - n_4 - d_1$ whereas user-2 travels distance value of 6 along the path $s_2 - n_5 - p''_1 - n_4 - d_2$. On the other hand, for MAX, a GTP query returns p'_1 from P. For user-1, the distance is 15 along the path $s_1 - p'_1 - n_2 - n_3 - d_1$. For user 2, the path is $s_2 - n_2 - p'_1 - n_2 - d_2$ resulting in travel distance value of 11.

The group can fix the order of visiting POIs visit or make the order flexible. For example, in a kGTP query, the group may fixed that they will first watch a movie then visit a shopping center or the the group may be happy to visit the movie theater and the shopping center in any order. We develop a solution that work for both *ordered and flexible kGTP queries.*

1.2 Solution Overview

Single user trip planning query has been extensively studied [Li05, Sha08, Che08, Ohs12], which do not take aggregate travel distance into account. The straightforward application of trip planning algorithm to retrieve trips for every group member independently and then selecting the trip with the minimum aggregate travel distance does not confirm the optimal answer. On the other hand, the proposed GTP algorithm [THK13] for the Euclidean space incurs very high query processing overhead even for computing the GTP answer in the Euclidean space for three categories of POIs. The processing of GTP queries in road networks is challenging because the distance computation is more complex in road networks than that of the Euclidean space. Therefore, we cannot extend the existing GTP algorithm in the Euclidean space for road networks. We develop a novel approach that can evaluate GTP queries in road networks with reduced query processing overhead.

The key idea of our approach is to retrieve an initial set of POIs, refine the search space based on the retrieved POIs and then identify the POIs within the refined region that minimize the aggregate distance for the group members. To refine the search space, we use elliptical properties. We compute an ellipse for each user in the group, where the foci of the ellipse is located at the user's source and destination and the major axis of the ellipse is equal to the aggregate trip distance computed based on the initially retrieved POIs. The refined search area is the intersection of all ellipses computed for the group members. According to the elliptical property, the distance between two foci of an ellipse via a point outside the ellipse is greater or equal to the length of the major axis. Thus, a POI outside the refined area can not contribute in minimizing the aggregate travel distance.

1.3 Contributions

In summary, the contributions of the thesis are as follows:

- We formalize group trip planning (GTP) queries in road networks. We present the first solution to process GTP queries in road networks for aggregate functions SUM and MAX.

- We exploit elliptical properties to refine the search region for the POIs and develop efficient algorithms to evaluate GTP queries.
- Our extensive experiments show that our proposed approach outperforms a naive approach in terms of both I/Os and computational time.

1.4 Thesis Organization

The next chapters are organized as follows. In Chapter 2, related works are discussed. In Chapter 3 and 4, we present our approach and a naive approach, respectively, to evaluate GTP queries. Chapter 5 shows the experimental results for our proposed algorithm. Finally, in Chapter 6, we conclude with future research directions.

Chapter 2

Related Works

In this chapter, we give a description of previous works related to this thesis. Query processing technique in spatial database has been extensively studied in recent years. These queries range from simple nearest neighbor (NN) queries to trip planning (TP) queries, focusing on both Euclidean and road networks. In this chapter, we first show the related works on NN queries in Section 2.1, aggregate nearest neighbor (ANN) queries in Section 2.2 Group nearest neighbor (GNN) queries in Section 2.3, Section 2.4 represents the related work on TP queries and finally GTP query and discussion on R-tree is given in Sections 2.5 and 2.6 respectively.

2.1 Nearest Neighbor Queries

NN queries and k nearest neighbor (kNN) queries constitute a very important category of queries in database studies. They have many kinds of applications including but not limited to geographic information systems GIS, CAD/CAM, multimedia [CFM94], knowledge discovery [MES99] and data mining. The first problem of answering kNN queries using the R-tree was first introduced in [NRV95]. That algorithm searches the R-tree in a depth first manner, using two different metrics to help pruning intermediate nodes and leaf nodes. One metric is optimistic mindist, which corresponds to the shortest distance between the query point and the MBR (Minimum Bounding Rectangle) of a tree entry. The other is pessimistic minmaxdist, which measures the longest distance from the query point to a tree entry MBR that ensures the existence of some data point (s). For a given query point q, mindist and minmaxdist are used to order and prune R-tree node entries according to three heuristics:

- Every MBR M with mindist (q,M) greater than the actual distance from q to a given object o is discarded;
- For two MBRs M and M', if mindist (q,M) is greater than minmaxdist (q,M'), M is pruned;
- If the distance from q to a given object o is greater than minmaxdist (q,M) for an MBR M, o is discarded.

The depth-first kNN algorithm accesses more R-tree nodes than necessary. To enable optimal index node access, a best-first algorithm was proposed in [HS99, PM97]. A memory heap is used to hold the R-tree entries to be searched, which gives priority to smaller mindist between an entry and the query point. Entries for the memory heap are selected according to the mindist between an entry and the query point. Only those entries with a small enough mindist are pushed into the heap and searched later with its sub-entries checked and pushed back if necessary. With the optimization done with the metric mindist, the best-first algorithm only accesses those nodes containing k nearest neighbors, thus achieving optimal node access.

Yu et al. [CYJ01] proposed an efficient method, called iDistance, for k nearest neighbor kNN search in a high-dimensional space. iDistance partitions the data and selects a reference point for each partition. The data in each cluster are transformed into a single dimensional space based on their similarity with respect to a reference point. This allows the points to be indexed using a B+-tree structure and kNN search be performed using one dimensional range search. The choice of partition and reference point provides the iDistance technique with degrees of freedom most other techniques do not have. They describe how appropriate choices here can effectively adapt the index structure to the data distribution.

The majority of the research before [CSS01] were based on Euclidean distance between the points. The first contribution of this paper is that it demonstrates Euclidean metric is not good distance approximation for road network. In that case they worked with real-world dataset. And they conclude that about 40% false hits are counted when Euclidean distance is used.

Shekhar et al. [SY03] introduced In-route nearest neighbor query, called IRNN. They discussed four different techniques namely Simple Graph-Based (SGB), Recursive Spatial Range query-based (RSR), Spatial Distance Join-based (SDJ) and Precomputed Zone-based (PCZ) solution. Precomputed zone-based method always outperforms the other methods when there are no updates on the road map. However, the response time of the precomputed zone-based method dramatically increases with increases in the update ratio I/Os. It also needs more storage space for storing the Precomputed results. In the other methods, the overall response time decreases with increases in the facility density and increases with increases in the route length and the size of the road map. The spatial distance join-based method outperforms the recursive methods with fewer numbers of path computations. The experiment shows that the strategy to reduce the number of path computations to minimize the response time is reasonable. However, in the dense road map area, the performance of the recursive spatial range query-based method declines due to the increase of the false hit ratio of the Euclidean candidates and the simple graph-based method shows better response time than the spatial range query-based method.

Conventional NN search (i.e., point queries) and its variations in low and high dimensional spaces have received considerable attention (e.g. [KF96, SK98] due to their applicability in domains such as content based retrieval and similarity search. Because of the proliferation of location based e-commerce and mobile computing, continuous nearest neighbor CNN search promises to gain similar importance in the research applications communities. Algorithm of CNN query first introcuced in [SZ01] in which the proposed solution has the false misses and the high processing cost. In [SK98] the study of the problem is done extensively and propose algorithms that avoid the pitfalls of previous ones, namely, the false misses and the high processing cost. They also propose theoretical bounds for the performance of CNN algorithms and experimentally verify that their proposed methods are nearly optimal in terms of node accesses.

However Kolahdouzan and Shahabi [KS04] introduced Voronoi-Based k Nearest Neighbor Search for Spatial Network Databases.

2.2 Aggregate Nearest Neighbor Queries

Papadias et al. [Pap05] have proposed and solved query processing in the context of large road network by extending kNN queries. They proposed three algorithms that solve ANN queries by effectively minimizing network traversal. The first one utilizes Euclidean lower bounds and an incremental Euclidean ANN method. The other two algorithms are motivated by aggregate top-k query processing techniques. The incremental Euclidean restriction IER incrementally retrieves Euclidean aggregate nearest neighbors and computes their network distances by shortest path queries until the result cannot be improved. TA and CE explore the network around the query points until the aggregate nearest neighbors are discovered. Their techniques can be applied for various aggregate distance functions (sum and max). In addition, they can be combined with spatial access methods and shortest path materialization techniques. A thorough experimental study suggests that their relative performance depends on the problem characteristics. IER is the best algorithm when the edge weights are proportional to their lengths since, in that case, Euclidean distance becomes a quite tight lower bound of the actual network distance. Nevertheless, the performance of IER degrades fast as the weights are less reflected by the edge lengths. For such cases, threshold algorithm TA is the most appropriate method for sum queries, whereas CE concurrent expansion is the best approach for max queries. In addition, TA and CE are the only choices when the interesting points are not indexed by R-trees or when the Euclidean distance bounds may not be used (e.g., in non spatial networks).

2.3 Group Nearest Neighbor Queries

A group nearest neighbor (GNN) query [DPM04, TH10, XL08] is slightly different form of NN query, finds the nearest data point with respect to all user locations of a particular group. A GNN query minimizes the aggregate distance for the group, where an aggregate distance is computed as the total, minimum or maximum distance of the data point from the group. Papadias et al. [DPM04] have proposed three techniques: multiple query method MQM, single point method SPM, and minimum bounding method MBM, to evaluate GNN queries.

- The multiple query method MQM is a threshold algorithm. It executes incremental NN search for each point q_i in Q, and combines their results. The distance to q_i's current NN is kept as a threshold t_i for each q_i. The sum of all thresholds is used as the total threshold T. best dist is dist (NN_{cur}, Q), where NN_{cur} is the candidate nearest neighbor found so far. Initially, best dist is set to 1 and T is set to 0. The algorithm computes the nearest neighbor for each query point incrementally, updating the thresholds and best dist until threshold T is larger than best dist.

- The single point method SPM processes a GNN query in a single traversal of the R-tree. SPM first decides the centroid q of Q, which is a point in the data space with a small or minimum value of dist (q, Q). Then a depth-first kNN as the query point. During the search, some heuristics based on triangular inequality is used to prune intermediate nodes and determine the real nearest neighbors to Q.

- The minimum bounding method MBM regards Q as a whole and uses its MBR M to prune the search space in a single query, in either a depth-first or best-first manner. Two pruning heuristics involving the distance from an intermediate node N to M or query points are proposed and can be used in either manner.

If we compare among the above three techniques MQM retrieves the NN for every point in query set Q, it sometimes accesses the same tree nodes more than once for different query points, so it causes the overhead while high query set cardinality is adopted. In contrast with MQM, both SPM and MBM perform a single query with some pruning heuristics. SPM is a modified single depth-first NN search with the centroid of Q being the query point while MBM considers the MBR of Q with best-first manner. The distribution shape of query points in Q has an impact on the performance of SPM and MBM because the former approximates q_i's centroid and uses it in the single query, and the latter takes into account the extent of Q when pruning nodes. According to the comparison conducted in [DPM04] among these three techniques, MBM performs the best as it traverses the R-tree once and takes the area covering the user's location into consideration.

Li et al. [FL11] have showed the MBM method only inspects the triangle inequality and there all other important pruning and optimization opportunities are failed to achieve. That is why they introduced a heuristic approach having query cost $O(N + M)$ according to the assumed facts. They first addressed a new exact method which has much lower query cost in compare to MBM method. They also explore high-quality approximation method that is bounded to logarithmic query cost and it has approximation ratio $\sqrt{2}$ in the worst case for a fixed dimension.

Huang et al. [HL05] have evolved two pruning strategies namely distance pruning method and MBR pruning method for GNN queries over spatial datasets indexed by the R-tree. By taking into account the distribution of all query points, we use an ellipse in both methods to approximate the query extent. Then a distance or MBR derived from the ellipse is used to prune intermediate index nodes during search. The proposed pruning strategies can be used in both the depth-first and best-first traversal paradigms. The experimental results demonstrate that their proposed methods outperform the existing ones significantly and consistently with real geographical datasets, in both page access number and CPU time.

2.4 Trip Planning Queries

Our problem has some similarities with TP query problem. Trip planning queries [Li05, Sha08, Che08, Ohs12] have been addressed in the research field which is exact and efficient solutions for the general optimal route queries. A set of point of interests POIs of different categories, starting point and destination point is given and TP query finds the best trip starting from the specific source and will ends on the destination through some POI with respect to a single user. On the other hand our problem GTP queries dealt with a number of users.

Cheng et al. [Li05] have defined the TP query problem which is $NP-hard$. So in that case they proposed approximation algorithms in terms of m, in terms of r and in terms of both m, r (where m represents the categories and r represents maximum category cardinality). They provide two greedy algorithms respectively for NN (nearest neighbor) algorithm with tight approximation ratios 2^{m+1} - 1 and MD (minimum distance) algorithm with approximation ratio $m + 1$ in terms of m. MD algorithm restricts the search space as an ellipse. Approximation algorithm in terms of maximum category cardinality, r use integer linear programming and approximation algorithm in terms of m and r use minimum spanning tree. They use their solution both in case of road network and Euclidean distance.

Sharifzadeh et al. [Sha08] have evolved algorithms to valuate an optimal sequenced route (OSR) query that returns a route with the minimum length passing across a set of data points in a particular order from the source location of a user, where both order and type of data points are given by the user. Chen et al. [Che08] have put a general trip planning query forward for consideration, called multi-rule partial sequenced route (MRPSR) query. A MRPSR query provides a homogeneous fundamental structure to valuate both of the above referred variants [Li05, Sha08] of trip planning queries. All these existing methods refer a single user, and so they are not fit for a group trip planning query. A large body of research work focus on developing algorithms for processing a route planning query and its variants [Gei10, She10, Mal11, Zho11]. A route planning query finds an acceptable route that minimizes the desire function such as travel time, shortest path, cost, etc. for a single source-destination pair. Route planning queries exclude data points in the route and are relevant for a single user. On the other hand, we propose an efficient solution that finds different types of data points for a group trip that reduce the total traveling distance at the least possible degree for the group.

2.5 Group Trip Planning Queries

Location-based social networks such as Facebook [fac], Google+ [goo], and Loopt [loo] have enabled a group of friends to remain connected virtually from anywhere at any time via location aware mobile devices. In this regard, GTP query problem [THK13] has been introduced for multiple sources and destination locations in Euclidean distance. In that purpose they presented two algorithms, among them one is iterative method as baseline algorithm and the other is an efficient hierarchical method. The iterative algorithm does not evaluate the trip planning query individually for every user but still have to access the same database more than once that causes more queries processing overhead. And on the other hand they evolved hierarchical algorithm that evaluates k-GTP queries with a single visit to the database and renders less processing overhead. They proposed hierarchical algorithm based on two heuristics, GTP-HA1 and GTP-HA2. Heuristic 1 and heuristic 2 are yielded to compute smaller upper bound of total distance. In terms of I/Os if the three algorithms are compared, iterative algorithm requires two orders of magnitude more I/Os than that of hierarchical algorithm. And GTP-HA1 takes on average 20% more I/Os and 27% more processing time than GTP-HA2.

In general when the users plan a trip that includes different types of data objects the number of different data points is typically restricted to 2 or 3. For example a group would like first go for shopping, taking dinner afterwards and possibly go to watch cinema later on. When a group will be planned for a large number of places and considered for flexible k-GTP queries then the query processing time may cause higher complexity. Let consider a group specify the order of visiting types {2,1,3} then in case of flexible group they also have to compute {3,1,2},{1,2,3}, {3,2,1},{1,3,2},{2,3,1}. So m>1 we have to consider m! visiting types for computing k-GTP queries. Thus they set the number of data types (m) to 2 and 3. They also provide the reason to consider ordered k-GTP queries because of a group wants to visit different places they know beforehand in which order they will visit the different places.

2.6 R-Tree: An efficient indexing Method

B-tree [PLL81] is a data structure for storing data with amortized run times for insertion and deletion in logarithmic time. It is often used for data storing on long latency I/Os (such as file systems and Data bases) because child nodes can be accessed together since they are in order. B-tree can not store new type of data (i.e. geometrical data, multi-dimensional data). So Guttman [Gut84] provided R-Tree to do that. R-tree [Sel87, Bec90] has some variants and the following properties in common-

- R-Trees can organize any-dimensional data by representing the data by a minimum bounding rectangle MBR.
- Each node bounds it's children. A node can have many objects in it.

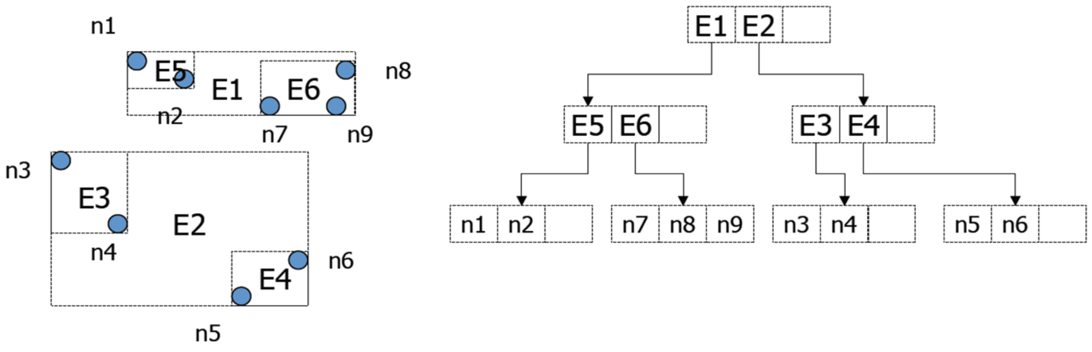


Figure 2.1: MBR corresponding to a R-tree

- The leaves point to the actual objects which are stored on the disk.
- The height is always log n (It is height balanced).

Chapter 3

Our Approach

In this chapter, we propose a novel approach to evaluate kGTP queries. In a kGTP query, the group provides the source and destination locations of the group members, k, and the required categories of POIs and the query returns k sets of POIs as answer. Road networks are enormous in size with numerous POIs and the operation on all POIs brings huge processing overhead. The underlying idea of our approach is to refine the search space to reduce the number of POIs to be considered. The steps of our solution are as summarized as follows:

Step 1: We develop a heuristic technique to retrieve an initial candidate answer set that includes one POI from each category. Then we compute the aggregate distance of the retrieved POIs with respect to the source-destination pairs of the group members.

Step 2: Based on the retrieved candidate answer set in the previous step, we refine the search region using elliptical properties. Any data point outside the refined search region cannot minimize the aggregate travel distance.

Step 3: This step executes a range query to find all POIs included in the refined region. From the retrieved POIs, our approach finds the actual POIs that provide the minimum aggregate travel distance.

In Sections 3.1, 3.2, and 3.3, we discuss detail of these three steps. Section 3.4 shows our algorithms and presents detail discussion. In Section 3.5, we propose our algorithm in a way that can solve flexible kGTP queries.

3.1 Selection of Initial POIs

In this section, we present a heuristic to retrieve an initial answer set that includes one POI from every requested category. Our heuristic is based on the intuition that the selection of a path for one user affects all others in the group. We first show how our heuristic works for a kGTP query, when the visiting order of POIs is fixed.

Our developed heuristic first computes geometric centroid, c_s and c_d, with respect to the set of source and destination locations of the group members, respectively. From c_s to c_d, we apply Nearest Neighbor (NN) search greedily for the next POI from the ordered categories. This initial answer set P contains a POI from each required category. Without loss of generality, Figure 3.1 shows an example, where a GTP query needs to be evaluated from 3 categories of POIs C_1, C_2, and C_3, $k = 1$ and for source and destination sets, $\{s_1, s_2, s_3\}$ and $\{d_1, d_2, d_3\}$, respectively. Our heuristic first finds nearest neighbor p_1 of c_s from C_1, then the nearest neighbor p_2 of p_1 from C_2, and finally, the nearest neighbor p_3 of p_2 from C_3.

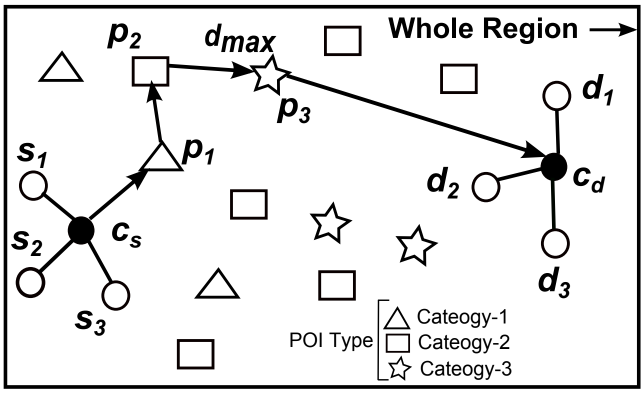


Figure 3.1: Centroid method

For each user u_i ($i = 1, 2, ..., n$), our approach computes the trip distance T_i from s_i to d_i via p_1, p_2, $\ldots p_m$. Then we apply the aggregate function f to determine the aggregate trip distance as d_{best} for the initial answer set, P_{dum}. To remind the reader, for SUM, the aggregate distance is $\sum_{i=1}^{n} T_i$ and for MAX, the aggregate distance is $\max_{i=1}^{n} T_i$.

For $k > 1$, our heuristic retrieves k nearest neighbors from C_m with respect to p^1_{m-1} as $p^1_m, p^2_m, \ldots p^k_m$. Our approach computes aggregate travel distances for k sets of POIs $\{p_1, p_2, ..., p^1_m\}$, $\{p_1, p_2, ..., p^2_m\}$, $\ldots$, $\{p_1, p_2, ..., p^k_m\}$ and initializes d^k_{best} with the maximum of these aggregate distances.

In a flexible $kGTP$ query, the order of visiting POI category is not fixed. The group only mentions the required POI categories and is happy to visit the POIs in any order that minimizes the aggregate travel distance. In a flexible $kGTP$ query, our approach randomly selects an order of visiting POI categories and then apply the same heuristic for the ordered kGTP query discussed above to retrieve the initial answer set P_{dum} for that order. After retrieving the initial POIs, our approach computes aggregate trip distances for all POI sets considering all possible orders and initializes d^k_{best} with the k^{th} minimum of these aggregate distances. For example, if the $P = \{p_1, p_2, p_3\}$ then the possible sets are $\{p_1, p_2, p_3\}$, $\{p_1, p_3, p_2\}$, $\{p_2, p_1, p_3\}$,$\{p_2, p_3, p_1\}$,$\{p_3, p_1, p_2\}$, and $\{p_3, p_2, p_1\}$.

3.2 Refinement of Search Region

We use the minimum distance d^k_{best} from step 1 to reduce the search space from the *whole search region* to a *refined search region* for both ordered and flexible kGTP queries. The whole road networks with numerous data points of various categories is the *whole search region.* The search for the solution in this whole region is infeasible. We refine the search space using d^k_{best} as follows:

We consider n ellipses $E_1, E_2, ..., E_n$ for each user $u_1, u_2, ..., u_n$. Ellipse E_i has foci at s_i and d_i, and length of the major axis equal to d^k_{best}. Let, E_{int} be the intersection region of the ellipses, i.e., $E_{int} = \cap^n_{i=1} E_i$. We confine the search only within E_{int}.

The following lemma and theorem show that the optimal answer resides within E_{int}.

LEMMA 1: $E_{int} \neq \emptyset$

Proof. Let, p be a data point in the k^{th} optimal solution found so far. For any arbitrary user u_l (where $1 \leq l \leq n$), $d(s_l, p) + d(p, d_l) < T_l$ where T_l is the trip distance of u_l in the k^{th} optimal solution. In case of sum aggregate function, $T_l < \sum^n_{i=1} T_i \leq d^k_{best}$, since d^k_{best} is the value of current best solution for SUM aggregate function. So, we find that $d(s_l, p) + d(p, d_l) < d^k_{best}$. This

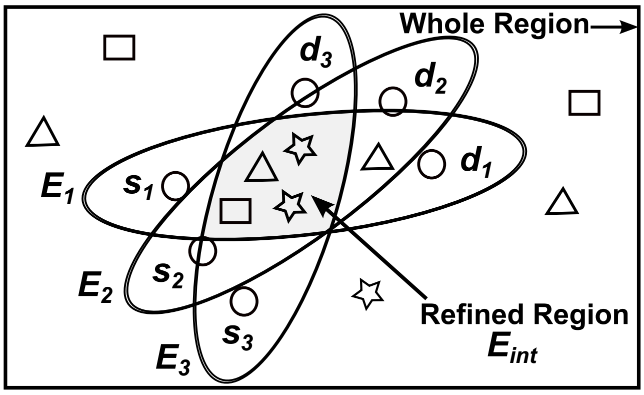


Figure 3.2: Refined search region

implies that point p resides inside the ellipse E_l (For any point inside an ellipse, the summation of distances from the point to the two foci is less than the major axis length). Since, u_l was chosen arbitrarily, point p must reside inside all ellipses $E_1, E_2, ..., E_n$. This means the intersection region E_{int} is non-empty.

Theorem 3.2.1 *The intersection of the regions E_{int} bounded by ellipses E_1, E_2, ..., E_n contains all the data points that constitute optimal solutions.*

Proof. (By contradiction). Let, p be a data point outside the region E_{int} and is a part of j^{th} optimal solution where $1 \leq j \leq k$. Since, p is outside E_{int}, it is outside of at least one of the ellipses $E_1, E_2, ..., E_n$. Let, E_l be such an ellipse that do not contain the point p where $1 \leq l \leq n$.

Since, user u_l passes through the data point p, the trip distance T_l in the j^{th} optimal solution cannot be less than $d(s_l, p)+d(p, d_l)$, i.e., $T_l \geq d(s_l, p)+d(p, d_l)$. According to the properties of ellipses, for any point outside of an ellipse, the summation of the distances from the point to the two foci is greater than its major axis length. Hence, for ellipse $E_l, d(s_l, p) + d(p, d_l) > d^k_{best}$. But, $T_l \geq d(s_l, p) + d(p, d_l)$. So, $T_l > d^k_{best}$.

Now, consider SUM aggregate function first. Then, total group distance for j^{th} solution will be greater than d^k_{best} as shown by $\sum_{i=1}^{n} T_i > T_l > d^k_{best}$.

But, d^k_{best} is the current best value for k^{th} optimal solution. So, j^{th} solution value cannot be greater than d^k_{best}. (Contradiction)

In case of MAX aggregate function, the maximum distance traveled by any user will also be greater than $d^k_{best} : max_{i=1}^{n} T_i > T_l > d^k_{best}$, however, d^k_{best} is the current best value of k^{th} solution. (Contradiction).

3.3 Optimal Answer Computation

In this step, our approach executes a range query on the POIs in the candidate answer set CA which resides inside the intersection of ellipses, E_{int}. The application of brute force technique to extract the solution increases the complexity with the increase of categories and users. We use d^k_{best} as current threshold for the aggregate trip distance and further prune the POIs, which makes the partial aggregate distance greater than d^k_{best}.

Specifically, we traverse starting from the source set S, pass through all categorized POIs in order and react at destination set D. If at any stage the cumulative distance greater than d^k_{best}, the traversal stops along the path. Each time a smaller cumulative distance appears by the traversal from source s_i to corresponding destination d_i through the ordered categorized POIs, we update d^k_{best}. Note that for flexible kGTP queries, we have to repeat the traversal of candidate POIs in all possible orders.

In Figure 3.3, let the predefined value for d^k_{best} is 19. Two categories C_1 and C_2 exist such that POIs $\{p^{'}_1, p^{''}_1, p^{'''}_1\} \in C_1$ and $\{p^{'}_2, p^{''}_2, p^{'''}_2\} \in C_2$. The traversal along path $s_1 - p^{'}_1 - p^{'}_2 - d_1$ is completely forsaken after visiting node $p^{'}_2$. Because at that node, the cumulative distance along that path is 20 which exceeds the range value 19. For another case, along the path $s_1 - p^{''}_1 - p^{''}_2 - d_1$, the cumulative distance becomes 15 at the destination point. This value is smaller than the predefined range d^k_{best}. So 15 becomes the new range value of d^k_{best}.

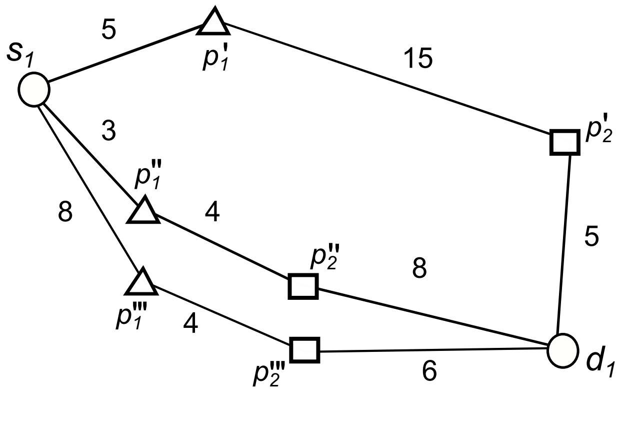


Figure 3.3: Optimal answer selection

3.4 Algorithms

Algorithm 1 shows the details steps for processing an ordered kGTP queries. The POIs are indexed using an R-tree in the database. The source location set S, the destination set D, the category set C, the aggregate function to be minimized f and k are the inputs. The output A is k sets of POIs that have k smallest trip distances.

Algorithm 1 starts with initializing the variables A, $dist$ and d^k_{best}. Then the algorithm applies our heuristic to retrieve the initial POIs from required categories (Lines 4–11). The algorithm uses function $FindCentroid$ to compute geometric centroid, c_s and c_d, with respect to the source and destination set, respectively. From c_s, the algorithm finds the greedy nearest neighbor y from category C_1 by using the function $FindNN$. The input to $FindNN$ are c_s (from which the NN will be computed), C_1 (from which category the NN will be computed) and 1 (the number of requested NN). Any existing nearest neighbor algorithm in road networks can be used for this purpose. P stores the the POIs computed using the proposed heuristic.

The function $ElementNum$ returns the number of categories in set C. Starting from the first category, $FindNN$ function returns the nearest neighbor of the next category with respect to the POI of the previous category, until the next category is C_{m-2}. All these neighbors are stored in P. Then the algorithm find k NNs from category C_m with respect to the identified POI p_{m-1}.

The algorithm uses the function $CalcBestD$ to compute k^{th} aggregate distance with respect to f and assigns it to the variable d^k_{best}. $CalcBestD$ takes S, D, P, k, f as inputs and computes k possible sets from POIs stored in P: $\{p_1, p_2, ..., p^1_m\}$, $\{p_1, p_2, ..., p^2_m\}$, $\ldots$, $\{p_1, p_2, ..., p^k_m\}$. $CalcBestD$ computes the aggregate trip distance for each set and returns the maximum one as k^{th} aggregate distance.

CalcBestD function is described in Algorithm 2. It takes in the parameters S, D, P, k, f and produces the value of d^k_{best} as output. To compute trip distances, we apply network distance function d for each user for each values of k. For each user, the cases considered are as follows: source to POI of category-1, POI of category-1 to orderered POIs in categories up to category-$(m-2)$, POI of category-$(m-1)$ to j^{th} POI of category-m, j^{th} POI of category-m to destination, where $j = 1, 2, \ldots k$. These are all added to produce total trip distance T_i for each user in Line 11. Then, if the case is for SUM, adding the maximum trip distances for each user gives the greedy best distance, shown in Line 15. Otherwise for MAX, MAX function in Line 17 gives the maximum value of trip distances for all users and the value is assigned to d^k_{best}.

In the second step, the algorithm executes *CandidateSet*, which takes three parameters S, D, d^k_{best} as inputs and returns all POISs in the refined search region as CA. More specifically, *CandidateSet* first computes the individual ellipses, compute the intersection of ellipses and then execute a range query to retrieve the POIs within the intersection of ellipses from the database.

After getting the candidate set CA, the algorithm searches for set of POIs k optimal trips have been found. The function *RangeQf* executes a search for k trips based on d^k_{best}. *RangeQf* maintains the following characteristics: the search towards a path terminates if the cumulative distance found so far exceeds d^k_{best} value. Each time an aggregate trip distance *dist* is smaller than d^k_{best}, d^k_{best} and A are updated.

3.5 Handling Flexible kGTP Queries

For flexible kGTP queries, as we have already mentioned in Sections 3.1 and 3.3, the algorithms need to determine paths by taking all possible orders of POI categories into consideration while computing distances d^k_{best} and *dist*. For other steps, the algorithm for flexible kGTP queries works in the similar way of ordered kGTP queries.

Algorithm 1 *kGTPQ*

```
Input: $S, D, C, f, k$
Output: $A$
1: $A \leftarrow$null, $P \leftarrow$null, $dist \leftarrow 0$, $d^k_{best} \leftarrow 0$, $i \leftarrow 1$, $m \leftarrow 0$
2: $c_s \leftarrow FindCentroid(S)$
3: $c_d \leftarrow FindCentroid(D)$
4: $y \leftarrow FindNN(c_s, C_1, 1)$
5: $P \leftarrow P \cup \{y\}$
6: $m \leftarrow ElementNum(C)$
7: while $i \leq$(m-2) do
8:     $x \leftarrow y$
9:     $y \leftarrow FindNN(x, C_{i+1}, 1)$
10:    $P \leftarrow P \cup \{y\}$
11:    $i \leftarrow i+1$
12: end while
13: $P \leftarrow P \cup FindNN(p_{m-1}, C_m, k)$
14: $d^k_{best} \leftarrow CalcBestD(S, D, P, k, f)$
15: $CA \leftarrow CandidateSet(S, D, d^k_{best})$
16: $i \leftarrow 1$
17: repeat
18:    for every $p^i_1$ in $C_1 \in CA$ do
19:        $dist \leftarrow RangeQf(S, D, CA, f, p^1_i, d^k_{best}))$
20:        if $dist < d^k_{best}$ then
21:            $d^k_{best} \leftarrow$dist
22:            $A \leftarrow A \cup RangeSet(S, D, p^i_1, d^k_{best})$
23:        end if
24:    end for
25: until All $k$ sets are found
```

Algorithm 2 *CalcBestD*

```
Input: $S, D, c_s, c_d, P, k, f$
Output: $d^k_{best}$
 1: $d^k_{best} \leftarrow 0$, $T \leftarrow null$, $i \leftarrow 1$, $j \leftarrow 1$, $m \leftarrow 0$, $n \leftarrow 0$, $r \leftarrow 1$
 2: $m \leftarrow ElementNum(P)$
 3: $n \leftarrow ElementNum(S)$
 4: for $i \leq n$ do
 5:    if $(m = 1)$ then
 6:       $p_{m-1} \leftarrow s_i$
 7:       $p_1 \leftarrow s_i$
 8:    end if
 9:    $j \leftarrow 1$
10:    for $j \leq k$ do
11:       $T^j_i \leftarrow d(s_i, p_1) + \sum_{r=1}^{m-2} d(p_r, p_{r+1}) + d(p_{m-1}, p^j_m) + d(p^j_m, d_i)$
12:    end for
13: end for
14: if $f$ denotes SUM then
15:    $d^k_{best} = \sum_{i=1}^{n}(max^k_{j=1}(T^j_i))$
16: else if $f$ denotes MAX then
17:    $d^k_{best} = max^{i \leq n, j \leq k}_{i=1, j=1}(T^j_i)$
18: end if
```

Chapter 4

Naive Solution

To compare the efficiency of our approach, we present a straightforward solution for processing kGTP queries. The naive approach is based on the search for answers in the total space without pruning any POI. The naive approach computes aggregate distances for all possible combination of ordered POI paths and identifies the k POI sets that provide the minimum aggregate answer set.

In Section 4.1, we give an overview of the naive solution. Section 4.2 presents the algorithm of the solution. In Subsections 4.2.1, 4.2.2, 4.2.3, and 4.2.4, we discuss the steps of the algorithm in detail. Finally Section 4.3 proposes the application of this algorithm to solve flexible kGTP queries.

4.1 Overview

We divide the key idea of the naive approach in four parts. In the first part, the aggregate distances from source set to each POI in the first category are computed. The second part computes the aggregate distances of all possible paths that passes through the POIs, where a path starts from a POI in the first category, moving through POIs from the intermediate ordered categories, reaching at a POI in the last category. In the third part of the algorithm, we compute the aggregate distances from destination set to each POI in the last category. The final part computes aggregate distances from the partial aggregate distances computed in the previous three parts for all possible paths from S to D via intermediary POIs. From these computed aggregate trip distances, the algorithm finds k POI sets that have k smallest values for the aggregate function

f. Figure 4.1 summarizes the process, where an example path is shown that starts from S, passes through $p_{11}, \ldots, p_{m1}$, and ends at D.

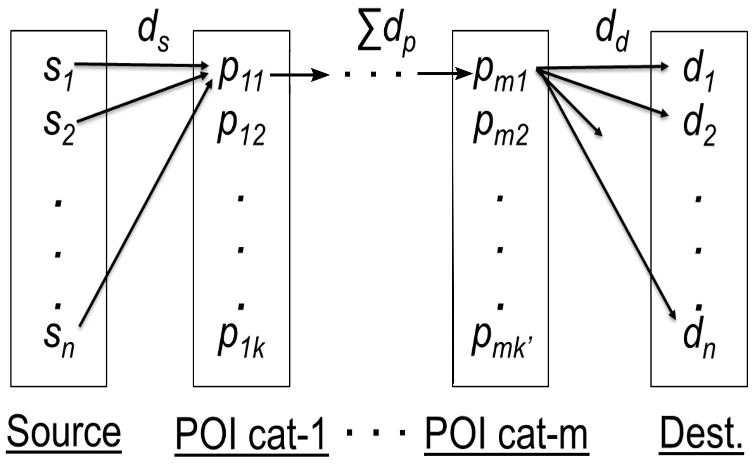


Figure 4.1: Naive approach

4.2 Algorithm

Algorithm 3 shows the pseudocode for the naive approach for processing ordered kGTP queries. The input to the algorithms are the source location set S, the destination set D, the category set C, the aggregate function to be minimized f and k. The output A is k sets of POIs that have k smallest trip distances.

We discuss the algorithm in the following four subsections.

4.2.1 d_s Formulation

First, we find the distance d_s from the source set S to every POI of the first category C_1. From Line 3, we see function $aggF$ takes in a set of sources S, a data point $p_i \in C_1$, aggregate function f. It gives the aggregate distance d_s as output considering f. If there are w POIs in category-1, then the algorithm uses $j = 1, 2, 3..., w$ in this loop and produces w number of d_s values.

Algorithm 3 *kGTPQ-NA*

Input: *S, D, C, f, k*

Output: *A*

```
1: $A \leftarrow$null, $dist_s \leftarrow$null, $dist_p \leftarrow$null, $dist_d \leftarrow$null, $dist_t \leftarrow null$, $i \leftarrow 1, j \leftarrow 1, r \leftarrow 1, t \leftarrow 1, m \leftarrow 0$
2: for every $p_1^j$ in $C_1 \in C$ do
3:     $dist_s(j) \leftarrow aggF(S, p_1^j, f)$
4: end for
5: $i \leftarrow 2, j \leftarrow 1$
6: $m \leftarrow ElementNum(C)$
7: while $i \leq m$ do
8:     for every $p_i^j$ in $C_i \in C$ do
9:         $dist_p(i) \leftarrow CombDist(p_i^j, CA, f)$
10:        $i \leftarrow i + 1$
11:    end for
12: end while
13: $j \leftarrow 1$
14: for every $p_m^j$ in $C_m \in C$ do
15:     $dist_d(i) \leftarrow aggF(D, p_m^j, f))$
16: end for
17: $i \leftarrow 1$
18: for every $dist_s(i)$ do
19:     $j \leftarrow 1$
20:     for every $dist_p(j)$ do
21:         $r \leftarrow 1$
22:         for every $dist_d(r)$ do
23:             $dist_t(t) \leftarrow sum(dist_s(i), dist_p(j), dist_d(m))$
24:             $A \leftarrow A \cup FindCandSet(S, D, dist_s(i), dist_p(j), dist_d(m), dist_t(t))$
25:         end for
26:     end for
27: end for
28: $FindBestk(A)$
```

4.2.2 $\sum d_p$ Formulation

For m categories, this step termed $\sum d_p$ formulation finds the distance in ordered path combinations from category-1 to category-m passing through all the intermediate $m-2$ categories. In Line 9, the function *CombDist* takes in a data point from C_1 category, the category set CA and aggregate function f as parameters. It computes and returns the distance from a combination by considering f. Here Dijkstra finds the path combinations from POIs of first category (i.e., C_1) to POIs of last category (i.e., C_m) by passing through all the intermediate ordered categories inside the function *CombDist*.

4.2.3 d_d Formulation

d_d is the aggregate distance from the POI of category m to the destination set D. In Line 15, the *aggF* function performs the task taking a set of destinations D, a data point $p_m^j \in C_m$ and the aggregate function f as parameters. It produces and aggregate distance value as output with respect to the D, p_m^j and f.

4.2.4 Exhaustive Search

For each ordered combination, the *sum* function adds the three distances formed before and the result is stored in $dist_t$. Line 23 in the algorithm shows this step. From Line 24, we find the candidate data point set using function $FindCandSet$. Finally from an exhaustive search done by the function $FindBestk$, we get the optimal result with the k^{th} smallest value for the aggregate function f.

4.3 Handling Flexible kGTP Queries

For flexible kGTP queries, the naive approach works in a similar fashion. The only difference is in the second step, where the naive approach for a flexible kGTP query requires to compute aggregate distances for paths considering all possible orders of POI categories. Thus, the number of candidate POI sets from which optimal answer set is identified increases for a flexible kGTP query in comparison with the ordered kGTP queries and causes increased query processing overhead.

Chapter 5

Experimental Study

5.1 Experiments

In this chapter, we evaluate the performance of our proposed range based kGTP query processing algorithm, $kGTPQ$, and compare it with the naive approach, $kGTPQ - NA$. We run all experiments using a computer with Intel Core i5 2.30 GHz CPU and 4GB RAM.

We have used both real and synthetic datasets to evaluate our solution. For real dataset, we have used road networks and point of interests (POIs) of Calfornia [Cal]. The road network has 21048 nodes and 21693 edges, and it contains 87635 POIs of 63 types of California. For synthetic data sets, we vary the size of the road networks and number of POIs (as shown in Table 5.2).

We have run our experiments for ordered kGTP queries, which is more common in real life. We have also set the default number of category as 3, as usually a group do not plan for more than 3 categories of POIs. Note that our algorithm is applicable for any number of categories.

In our experiments, we vary the following query parameters: (i) the group size n, (ii) the number of answer set of data points k, (iii) the query area i.e., the minimum bounding rectangle covering the source and destination locations M, and (iv) the size of data sets (synthetic data set). Table 5.1 summarizes the parameter values used in our experiments. In all experiments, we estimate I/Os and the query processing time to measure the efficiency of our algorithms. In each set of experiments, we run the experiment for 100 queries and present the average result.

Parameter	Values	Default
Group size	2, 4, 8, 16	4
Query area M	2%, 4%, 8%, 16%	2%
k	2, 4, 8, 16	8
Synthetic data set size	5K, 10K, 15K, 20K	-

Table 5.1: Values of different query parameters used in our experiments

Nodes	Edges	POIs	Types
5000	38354	57280	4
1000	65707	98424	4
15000	86567	129688	4
20000	107090	160732	4

Table 5.2: Parameters of synthetic datasets

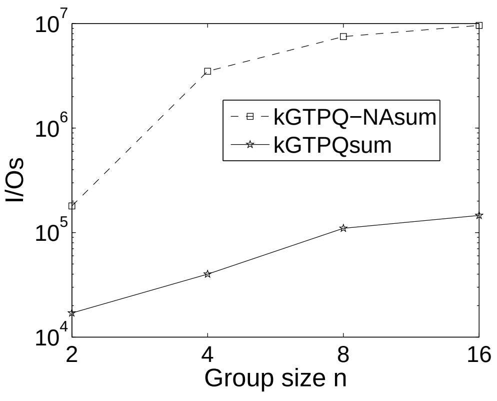
We first present our experiment results for processing kGTP queries for aggregate function SUM in Section 5.1.1, then we present experimental results for aggregate function MAX (Section 5.1.2). Finally, in Section 5.1.3, we compare the results of aggregate functions SUM and MAX.

5.1.1 Aggregate function SUM

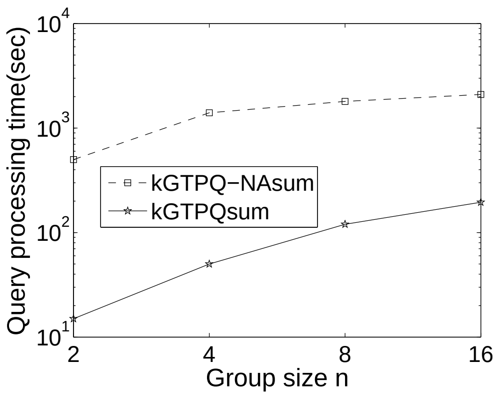
In this section, we present the experimental results for processing kGTP queries for aggregate function SUM. For SUM, we name our proposed approach and the naive approach as $kGTPQ_{sum}$ and $kGTPQ - NA_{sum}$, respectively. In the following sets of experiments, we vary group size, the answer set size, query area, and dataset size. When we vary one parameter, we set other parameters to default values as stated in Table 5.1.

Effect of group size n***:*** Figure 5.1(a) and 5.1(b) show I/Os and processing time, respectively, for processing kGTP queries using $kGTPQ_{sum}$ and $kGTPQ - NA_{sum}$. We observe that for both algorithms $kGTPQ_{sum}$ and $kGTPQ - NA_{sum}$, I/Os and processing time increase with the increase of the group size n. We also observe that in terms of I/Os, our approach $kGTPQ_{sum}$ outperforms the naive approach $kGTPQ - NA_{sum}$ by 2 orders of magnitude.

Moreover, the query processing time of $kGTPQ-NA_{sum}$ is on average one and half order of magnitude higher than that of $kGTPQ_{sum}$.



(a)



(b)

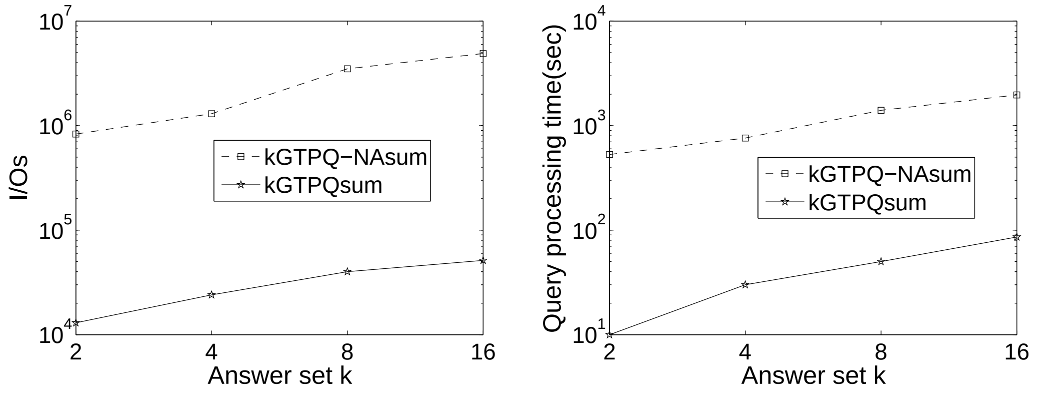
Figure 5.1: Effect of group size n for California data (a) I/Os and (b) query processing time

Effect of answer set k**:** In this set of experiments, we vary k as 2, 4, 8, and 16, and compare the experimental results $kGTPQ_{sum}$ and $kGTPQ-NA_{sum}$.

From Figure 5.2(a) and Figure 5.2(b), we observe that I/Os and processing time increase with the increase of k for both $kGTPQ_{sum}$ and $kGTPQ-NA_{sum}$. Figures show that $kGTPQ-NA_{sum}$ takes almost one and half orders of magnitude more I/Os compared to $kGTPQ_{sum}$. We also see from Figure 5.2(b) that the query processing time of $kGTPQ-NA_{sum}$ is on average one and half orders of magnitude higher than that of $kGTPQ_{sum}$. The superiority of our approach over the naive approach comes from the ellipse based pruning strategies that reduces the search space significantly.

Effect of Query Area M**:** In this case, we vary the query area M as 2%, 4%, 8% and 16% of the data space. Figure 5.3(a) and 5.3(b) show I/Os and query processing time, required by $kGTPQ_{sum}$ and $kGTPQ-NA_{sum}$, respectively. Since we need to access more data points from R-trees of different data types for a larger M, both I/Os and processing time increase with the increase of M for both approaches.

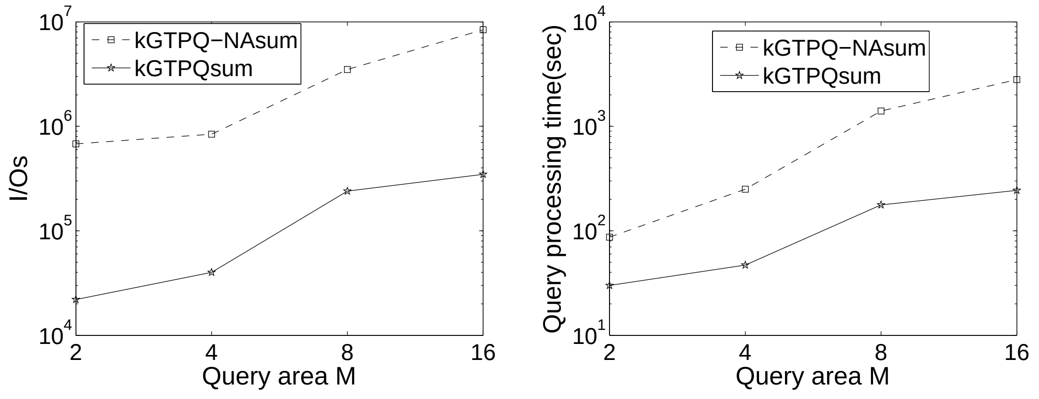
Figure 5.3(a) shows that $kGTPQ-NA_{sum}$ requires at least one and half



(a) (b)

Figure 5.2: Effect of answer set k for California data (a) I/Os and (b) Query processing time

orders of magnitude more I/Os than that of $kGTPQ_{sum}$. Similarly, Figure 5.3(b) shows that the processing time of $kGTPQ - NA_{sum}$ is on average one order of magnitude higher than that of $kGTPQ_{sum}$.

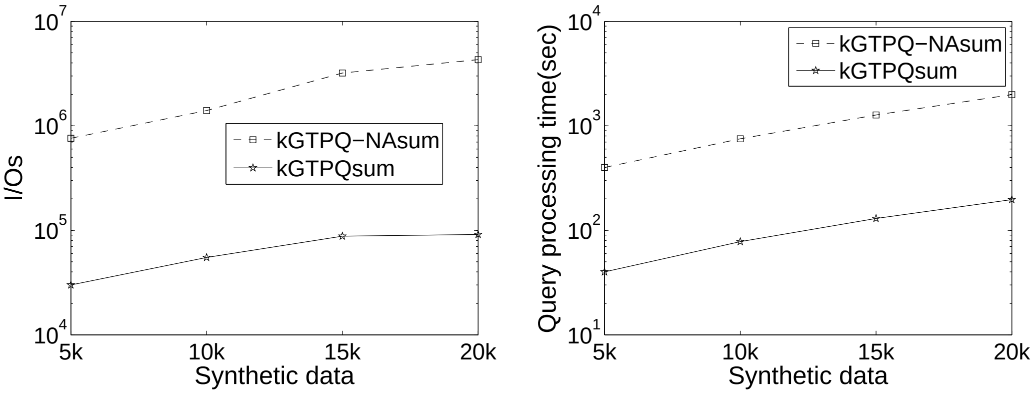


(a) (b)

Figure 5.3: Effect of query area M for California data (a) I/Os and (b) query processing time

Effect of dataset size: In this set of experiments, we vary the road network size and number of POIs as stated in Table 5.2. In this case, we use the first column, i.e., varying the number of nodes as 5000, 10000,15000 and 20000 to identify four synthetic datasets. Figure 5.4(a) and 5.4(b) show I/Os and processing time respectively for different synthetic dataset sizes. As expected, the experimental results show that $kGTPQ_{sum}$ outperforms $kGTPQ - NA_{sum}$

in a greater margin for a larger dataset size in terms of both I/Os and query processing time. Figure 5.4(a) shows the I/Os of $kGTPQ-NA_{sum}$ is at least on average one and half orders of magnitude higher than that of $kGTPQ_{sum}$ and from Figure 5.4(b), we see that the query processing time of $kGTPQ-NA_{sum}$ is on average one order of magnitude higher than that of $kGTPQ_{sum}$.



(a) (b)

Figure 5.4: Effect of varying dataset size (a) I/Os and (b) query processing time

5.1.2 Aggregate function MAX

In this section, we present the experimental results for processing kGTP queries for aggregate function MAX. For MAX, we name our proposed approach and the naive approach as $kGTPQ_{max}$ and $kGTPQ-NA_{max}$, respectively. In the following sets of experiments, we vary group size, the answer set size, query area, and dataset size, and compare the results of our approach with the naive approach.

Effect of group size n***:*** Figure 5.5(a) and 5.5(b) show I/Os and query processing times, respectively, for processing kGTP queries using $kGTPQ_{max}$ and $kGTPQ-NA_{max}$. For $kGTPQ-NA_{max}$, I/Os and query processing time almost remain constant with the increase of n. On the other hand for $kGTPQ_{max}$, I/Os and processing time slightly increase with the increase of group size from 4 to 8, and then it almost remains constant for a larger group. As expected, I/Os and query processing time of $kGTPQ_{max}$ are much less than those of $kGTPQ-NA_{max}$. Figure 5.5(a) and (b) show that I/Os and query processing time of $kGTPQ_{max}$ are on average one and half orders of magnitude smaller than those of $kGTPQ - NA_{max}$.

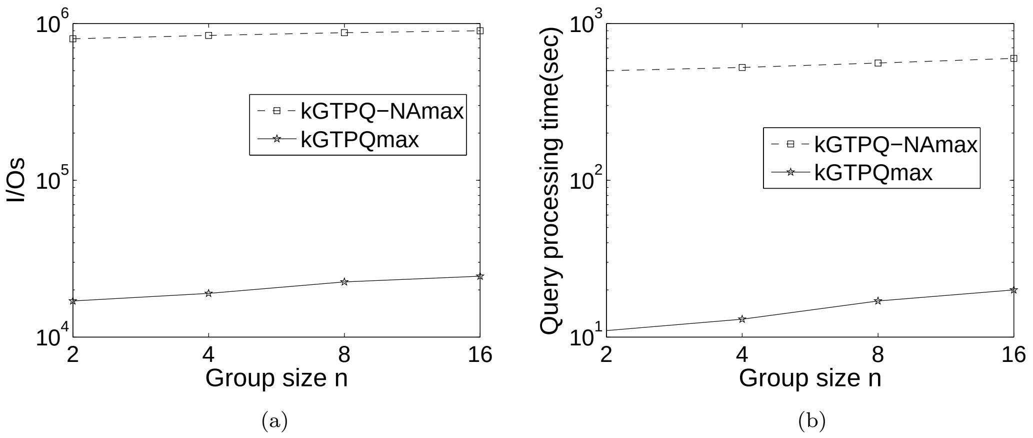


Figure 5.5: Effect of group size n for California data (a) I/Os and (b) Query processing time)

Effect of answer set k***:*** In this set of experiments, we vary k as 2, 4, 8, and 16, and compare the experimental results $kGTPQ_{max}$ and $kGTPQ - NA_{max}$.

In Figure 5.6(a), we observe that I/Os almost remain constant with the increase of k. We find that $kGTPQ - NA_{max}$ takes almost two orders of magnitude more I/Os compare to $kGTPQ_{max}$. Figure 5.6(b) shows that the query processing time of $kGTPQ - NA_{max}$ is at least one and half order of magnitude higher than that of $kGTPQ_{max}$ because it facilitates ellipse based pruning to reduce the search space significantly.

Effect of query area M***:*** In this set of experiments, we vary the query area M as 2%,4%,8%,16% of entire data space. Figure 5.7(a) and 5.7(b) show that I/Os and processing time, respectively, for $kGTPQ_{max}$ and $kGTPQ - NA_{max}$. We observe that I/Os and query processing time increase with the increase of M for both algorithms as for a larger M both approaches need to access more data points from R-trees than those of a smaller M.

Figure 5.7(a) shows that $kGTPQ - NA_{max}$ requires at least two orders of magnitude more I/Os than that of $kGTPQ_{max}$. Similarly, Figure 5.7(b) shows that the query processing time of $kGTPQ - NA_{max}$ is on average one and half

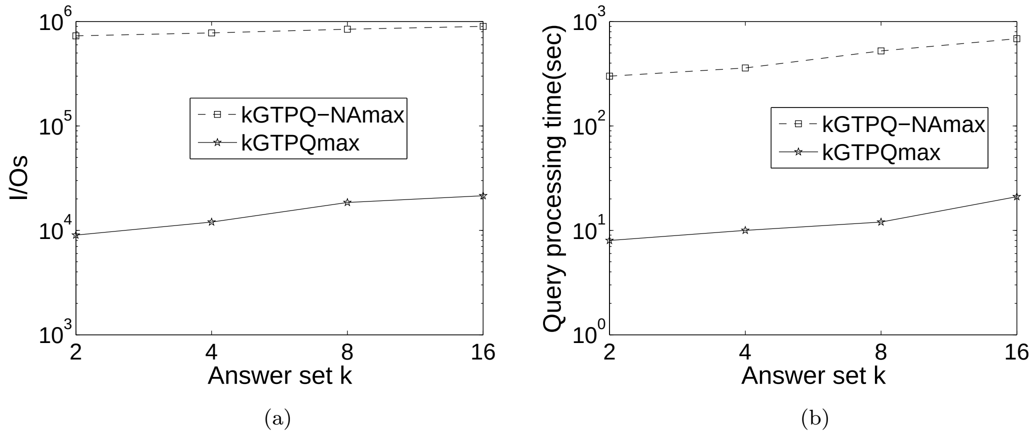


Figure 5.6: Effect of answer set k for California data (a) I/Os and (b) Query processing time

orders of magnitude higher than that of $kGTPQ_{max}$.

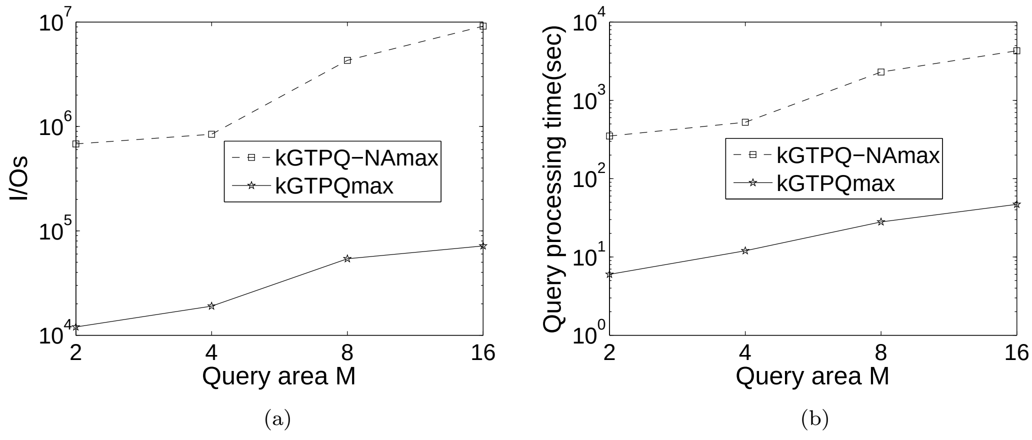


Figure 5.7: Effect of Query Area M for California data (a) I/Os and (b) Query processing time

Effect of dataset sizes: Similar to SUM, In this set of experiments, we vary the road network size and number of POIs as stated in Table 5.2.

Figure 5.8(a) and 5.8(b) show I/Os and processing time, respectively, for different dataset sizes. The experimental results show that $kGTPQ_{max}$ outperforms $kGTPQ - NA_{max}$ for all dataset sizes in terms of both I/Os and query processing time.

Figure 5.7(a) shows that I/Os of $kGTPQ - NA_{max}$ is on average one and half orders of magnitude higher than that of $kGTPQ_{max}$. Figure 5.7(b) shows that query processing time of $kGTPQ - NA_{max}$ is on average one and half orders of magnitude higher than that of $kGTPQ_{max}$.

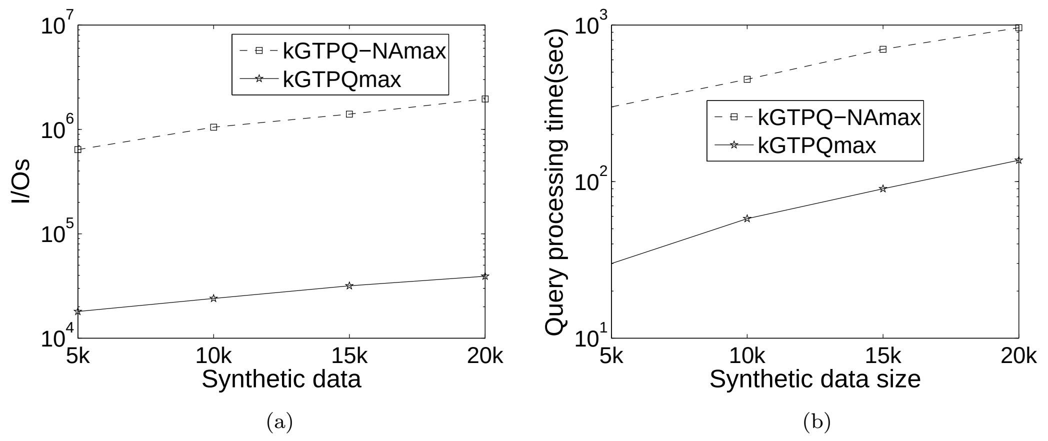


Figure 5.8: Effect of synthetic data set (a) I/Os and (b) Query processing time

Parameter	$kGTPQ_{sum}$	$kGTPQ_{max}$
Group size n	78225	20737
Query area M	162250	39198
k	32086	15240
Synthetic data set	66080	28251

Table 5.3: Average I/Os for $kGTPQ_{sum}$ and $kGTPQ_{max}$ for different parameters

Parameter	$kGTPQ_{sum}$	$kGTPQ_{max}$
Group size n	95s	15.25s
Query area M	102.13s	23.29s
k	44.03s	12.76s
Synthetic data set	111.21s	78.37s

Table 5.4: Average query processing time for $kGTPQ_{sum}$ and $kGTPQ_{max}$ for different parameters

5.1.3 Comparison between SUM and MAX

In this section, we compare I/Os and processing time of kGTPQ queries for aggregate functions SUM and MAX. Tables 3 and 4 summarize I/Os and query processing time, respective, by varying different parameters.

We observe that $kGTPQ_{sum}$ requires much higher I/Os and processing time than $kGTPQ_{max}$. The underlying reason is as follows. $kGTPQ_{max}$ considers the maximum distance of any user while $kGTPQ_{sum}$ considers the total distance of all users. As the total distance of all users is larger than the maximum distance of any user, the area of the pruning ellipse for $kGTPQ_{sum}$ is much larger than that of $kGTPQ_{max}$. Hence, $kGTPQ_{sum}$ accesses higher number of R-tree nodes than $kGTPQ_{max}$.

We have also done the experiments for $m = 2$. We omit the experimental results for $m = 2$ as the behavior is similar to what we have described in previous sections for $m = 3$. However, the processing time and I/Os for $m = 3$ are higher than that of $m = 2$, which is expected. Since in real scenarios, the number of data point categories is typically limited to 2 or 3, the trend of increasing performance overhead with an increase of m would not effect the applicability of our algorithms.

Chapter 6

Conclusion

In this thesis, we introduced a novel approach for processing k group trip planing (kGTP) queries in road networks. To the best of our knowledge, this work is the first to address kGTP queries in road networks. We focused on both aggregate functions, SUM and MAX. We have developed an efficient pruning technique to refine the search space by exploiting elliptical properties and based on the pruning technique, we proposed an algorithm to evaluate kGTP queries. Experimental results show that our approach outperforms a naive technique with a large margin both in terms of I/Os and computational overhead.

6.1 Future Directions

Our thesis work can be extended to several interesting directions in future. Some noteworthy future directions for this research are proposed below:

- In future, we aim to work for protecting location privacy of users while processing kGTP queries.
- In this thesis, we have considered ordered point of interest (POI) types. In future we plan to work with flexible order of POI types. It will be our future challenge.

References

[Bec90] Kriegel H.P. Schneider R. Seeger B Beckmann, N. The r*-tree: An efficient and robust access method for points and rectangles. *In SIGMOND*, 1990.

[Cal] Real data: http://www.cs.fsu.edu/ lifeifei/tpq.html.

[CFM94] M. Ranganathan C. Faloutsos and Y. Manolopoulos. Fast subsequence matching in time-series databases. *In Proc. of ACM SIGMOD Intl Conference*, 1994.

[Che08] Ku W.S. Sun M.T. Zimmermann R Chen, H. The multi-rule partial sequenced route query. *GIS '16th ACM SIGSPATIAL international conference*, pages 10:1–10:10, 2008.

[CSS01] M.R. Kolahdouzan C. Shahabi and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *In Proc. ACM Intl Workshop Geographic Information Systems*, 2001.

[CYJ01] K.-L. Tan C. Yu, B. Ooi and H. V. Jagadish. Indexing the distance: An efficient method to knn processing. *In Proc. of Very Large Data Bases Conference (VLDB)*, 2001.

[DPM04] Y. Tao D. Papadias, Q. Shen and K. Mouratidis. Group nearest neighbor queries. *In IDME*, pages 301–312, 2004.

[fac] Facbook: http://www.facebook.com.

[FL11] Kumar P Feifei Li, Bin Yao. Group enclosing queries. *In IEEE Transactions on Knowledge and Data Engineering*, 23(10):1526–1540, 2011.

[Gei10] Kobitzsch M. Sanders P Geisberger, R. Route planning with flexible objective functions. *In ALENEX*, pages 124–137, 2010.

[goo] Google+: http://plus.google.com.

[Gut84] A. Guttman. A dynamic index structure for spatial searching. *In SIGMOND*, pages 47–57, 1984.

[HL05] Bo Huang Zhiyong Huang Hongga Li, Hua Lu. Two ellipse-based pruning methods for group nearest neighbor queries. *GIS '05 Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 192–199, 2005.

[HS99] G. Hjaltason and H. Samet. Distance browsing in spatial database. *ACM Trans. on Database Systems*, 24(2), 1999.

[KF96] Faloutsos C.Siegel E Protopapas Z. Korn F., Sidiropoulos N. Fast nearest neighbor search in medical image databases. *In Proc. Very Large Data Bases Conference(VLDB)*, 1996.

[KS04] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. *VLDB conference*, 2004.

[Li05] Cheng D. Hadjieleftheriou M. Kollios G. Teng S.H Li, F. On trip planning queries in spatial databases. *In SSDL*, pages 273–290, 2005.

[loo] loopt: http://www.loopt.com.

[Mal11] Madden S. Bhattacharya A Malviya, N. A continuous query system for dynamic route planning. *In ICDE*, pages 792–803, 2011.

[MES99] H.-P. Kriegel M. Ester and J. Sander. Knowledge discovery in spatial databases. *Invited paper at German Conf. On Artificial Intelligence*, 1999.

[NRV95] S. Kelley N. Roussopoulos and F. Vincent. Nearest neighbor queries. *In Proc. of ACM SIGMOD Intl Conference*, 1995.

[Ohs12] Htoo H. Sonehara N. Sakauchi M Ohsawa, Y. Sequenced route query in road network distance based on incremental euclidean restriction. *Geoinformatica*, 15(3):484–491, 2012.

[Pap05] Tao Y. Mouratidis K. Hui C.K Papadias, D. Aggregate nearest neighbor queries in spatial databases. *In IEEE,ACM Transactions on Database Systems (TODS)*, 30(2):529–576, 2005.

[PLL81] S. BING YAO. PHILIP L. LEHMAN. Efficient locking for concurrent operations on b-trees. *IN ACM INTERNATIONAL CONFERENCE*, 1981.

[PM97] A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in r-trees. *In Proc. of Int l Conf. on Database Theory (ICDT)*, 1997.

[Sel87] Roussopoulos N. Faloutsos C. Sellis, T. The r+-tree: a dynamic index for multidimensional objects. *In VLDB*, 1987.

[Sha08] Kolahdouzan M. Shahabi C Sharifzadeh, M. The optimal sequenced route query. *International Journal on Very Large Data Bases*, 17(2):765–787, 2008.

[She10] Chen Z. Zhou X. Zheng Y. Xie X Shen, H.T. Searching trajectories by locations: an effciency study. *ACM SIGMOD International Conference on Management of data*, pages 255–266, 2010.

[SK98] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. *In Proc. of ACM SIGMOD Intl Conference*, 1998.

[SY03] S. Shekhar and J.S. Yoo. In-route nearest neighbor queries: A comparison of alternative approaches. *Proc. ACM Intl Workshop Geographic Information Systems*, 2003.

[SZ01] Roussopoulos N. Song Z. K-nearest neighbor search for moving query point. *SSTD*, 2001.

[TH10] Rui Zhang Tanzima Hashem, Lars Kulik. Privacy preserving group nearest neighbor queries. *In ACM SIG-SPATIAL*, 2010.

[THK13] Mohammed Eunus Ali Tanzima Hashem, Tahrima Hashem and Lars Kulik. Group trip planning queries in spatial databases. In *SSTD'13*, pages 259–276, 2013.

[XL08] Lei Chen Xiang Lian. Probabilistic group nearest neighbor queries in uncertain databases. *In IEEE*, 20(6), 2008.

[Zho11] Chen Z. Shen H.T. Zhou, X. Discovering popular routes from trajectories. *ICDE 'IEEE 27th International Conference on Data Engineering*, pages 900–911, 2011.