B.Sc. in Computer Science and Engineering Thesis

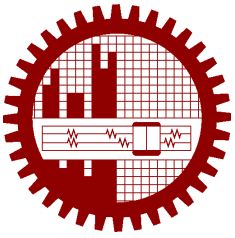
Efficient Scheduling of Generalized Group Trips in Road Networks

Submitted by

Yeasir Rayhan
1305111

Supervised by

Dr. Tanzima Hashem



Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

October 2018

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Efficient Scheduling of Generalized Group Trips in Road Networks", is the outcome of the investigation and research carried out by us under the supervision of Dr. Tanzima Hashem.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Yeasir Rayhan
1305111

CERTIFICATION

This thesis titled, **"Efficient Scheduling of Generalized Group Trips in Road Networks"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in October 2018.

Group Members:

Yeasir Rayhan

Supervisor:

Dr. Tanzima Hashem
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

Dhaka
October 2018

Yeasir Rayhan

Contents

List of Figures

List of Tables

ABSTRACT

In this thesis, we introduce generalized group trip scheduling (GGTS) queries that enable friends and families to perform activities at different points of interests (POIs) like a shopping center, a restaurant and a pharmacy with the minimum total travel distance. Trip planning and scheduling for groups, an important class of location-based services (LBSs), have recently received attention from the researchers. However, both group trip planning (GTP) and group trip scheduling (GTS) queries have restrictions: a GTP query assumes that all group members visit all required POIs together whereas a GTS query requires that each POI is visited by a single group member. A GGTS query overcomes these limitations and allows any specified number of group members to visit the same POI of the required type together. We develop the first comprehensive solution for processing GGTS queries in road networks. We develop an efficient algorithm to evaluate the exact answers for GGTS queries. Since finding the answer for a GGTS query is a NP-hard problem, to further reduce the processing overhead for a large group size, a large number of required POI types or a large POI dataset, we propose two heuristic solutions, trip scheduling heuristic (TSH) and search space refinement heuristic (SRH), for processing GGTS queries. Extensive experiments with real datasets show that our optimal algorithm is preferable for small parameter settings and the heuristic solution reduces the processing overhead significantly in return of sacrificing the accuracy slightly.

Chapter 1

Introduction

Efficient processing of group trip planning (GTP) [1] and group trip scheduling (GTS) [2] queries, a novel class of group centric location-based services, have received attentions from the researchers in recent years. GTP and GTS queries have applications in tour planning, event organization and home management with limitations. A GTP query enables a group to plan a trip with the minimum aggregate travel distance for visiting multiple points of interests (POIs) like a restaurant, a shopping center, and a movie theater *together*. On the other hand, a GTS query assumes that each member visits a number of POIs *independently* such that the aggregate travel distance of the group members is minimized. However, in real life scenarios POIs may need to be visited by more than one group member but not by all members; for example, at least two members of a family may need to visit a shopping center to choose a gift item or a restaurant to decide about a menu for organizing an event. Neither a GTP query nor a GTS query allows such flexibility in visiting POIs. To address this scenario, in this thesis, we introduce a novel query type, a *Generalized Group Trip Scheduling (GGTS) query*, where group members may visit a POI *together* or in a *sub-group* or *alone*.

To perform daily life activities, people usually have to visit fixed locations like office, friends' places and flexible locations like an ATM booth of a specific bank, any branch of a supermarket or a grocery shop etc. Given a set of fixed source-destination pairs for each group member, the required POI types for the group and the numbers of members who need to visit each POI type, a GGTS query returns a set of POIs for each member of the group such that all required POI types are visited by the specified number of group members and the aggregate trip overhead distance is minimized. Each member's trip starts from her source, then visits the returned POIs by the GGTS query and ends at her destination location. The trip overhead distance of a member is the additional distance that the member needs to travel for visiting the POIs (i.e., the trip distance minus the distance between source and destination locations) and the aggregate trip overhead distance of a group is measured as the summation of the trip overhead distances of all members in the group. In this thesis, we propose the first solution for processing GGTS queries in road

networks.

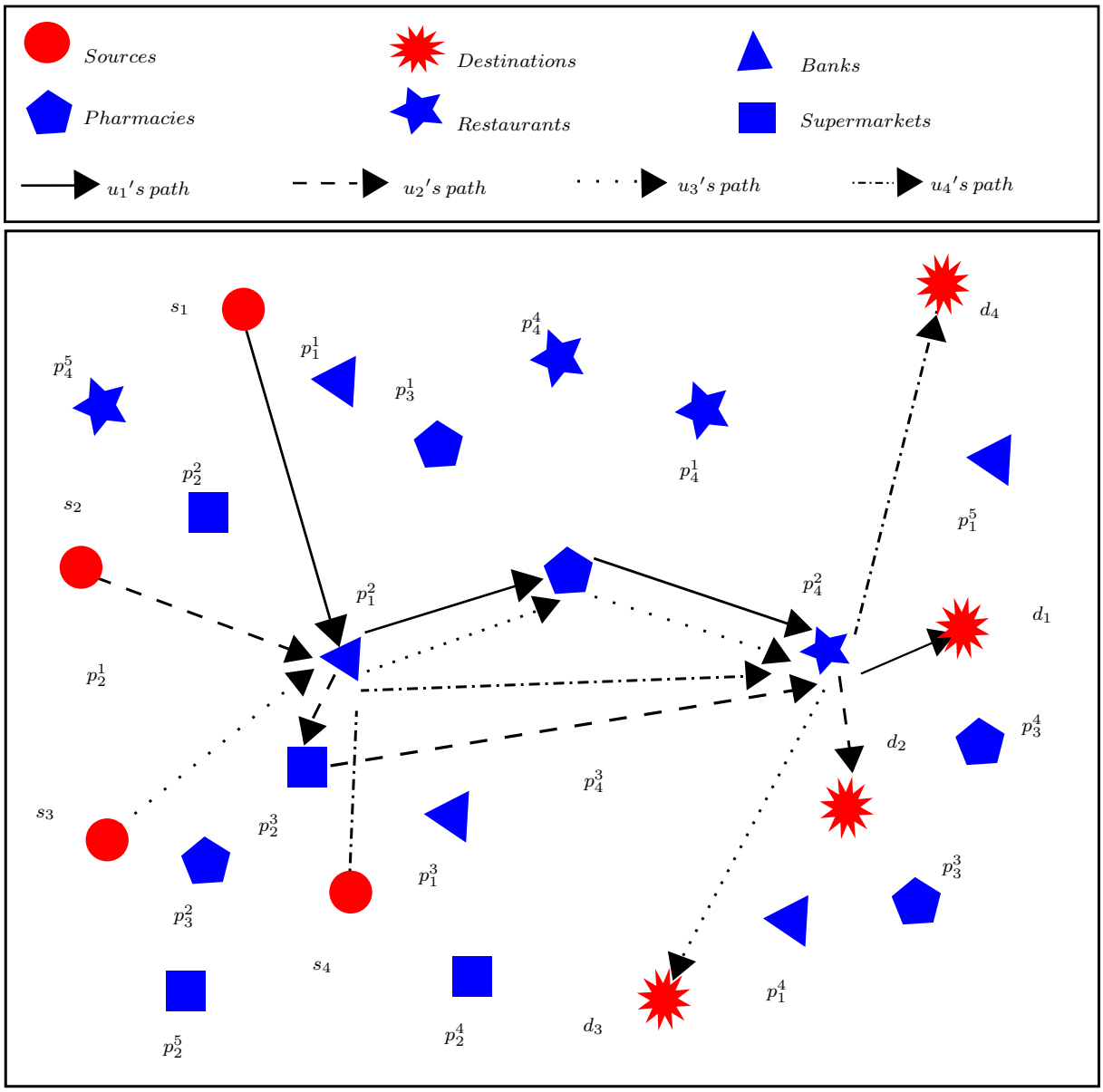


Figure 1.1: An example of a GGTS query

Figure 1.1 shows an example scenario of a GGTS query, where a group of four members would like to visit four POI types: a bank (c_1), a supermarket (c_2), a pharmacy (c_3), and a restaurant (c_4). Group members u_1, u_2, u_3, and u_4 have source destination pairs, $\langle s_1, d_1 \rangle$, $\langle s_2, d_2 \rangle$, $\langle s_3, d_3 \rangle$, and $\langle s_4, d_4 \rangle$, respectively. Here, $p_{c_j}^k$ denotes a POI of type c_j with ID k. For example, POI p_1^1 in the figure is of type 1, which represents a bank. First, all group members will go to a bank from their office so that one of the members can withdraw money from the bank and distribute it among themselves. Then, two of them will go to a supermarket for doing shopping, one of them will go to a pharmacy for buying medicines and one of them will directly go to a restaurant to order food in advance. At the end, other three members will join the member at the restaurant for having the dinner together. Finally, all members will return to their homes. Note that when a POI type is expected to be visited by more than one group members, then all members will visit

the same POI (e.g., same branch of a bank). For each POI type, there are several options (e.g., there can be many branches of a bank) to choose from. A GGTS query considers all options for every POI type, and returns four trips for four group members with the minimum aggregate trip overhead distance. Figure 1.1 shows four scheduled trips, where four trips together visit all required POI types, i.e., a bank, a supermarket, a restaurant, and a hospital.

The challenge for computing a trip in real time comes from finding the POI set that minimizes the travel distance from a huge number of possible POI sets. The challenge is further intensified if multiple trips are computed for a group, specially if any number of group members can visit the same POI like in GGTS queries. The processing overhead of our solution for a GGTS query depends on the considered POI search space and the efficiency of the scheduling algorithm. Though POI search space refinement techniques are commonly used for reducing a query processing overhead [1–4], the way the POI search space is refined differ for variant query types. We exploit geometric properties and refine the POI search space for GGTS queries. We prove that the set of POIs that provide the minimum aggregate trip overhead distance are located inside the refined POI search space. The smaller the refined POI search space, the smaller is the number of possible POI sets considered for finding the optimal answer. Then we develop an efficient trip scheduling algorithm for computing trips for GGTS queries using all candidate POIs located in the refined POI search space. Note that the existing dynamic programming techniques [1, 2] for scheduling trips for GTP and GTS queries are not adaptable for GGTS queries because a GGTS query allow any number of group members' visit to a same POI.

Trip computation problem is shown as an NP-hard problem in the literature [5], which means the problem of evaluating GGTS queries is also NP-hard as it requires computation of trips for the group members. When the group size or the required number of POI types or the POI dataset is large, to further reduce the processing overhead, we develop two heuristic solutions for GGTS queries: trip scheduling heuristic (TSH) and search space refinement heuristic (SRH). TSH greedily schedules the trips for the group members using the candidate POIs that include the optimal answer, whereas SRH reduces the number of candidate POIs by approximating the POI search space and uses the same trip scheduling algorithm as the optimal solution.

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

- We introduce and formulate a new type of query, the generalized group trip scheduling (GGTS) query in road networks.
- We present an efficient solution for finding the optimal answer for GGTS queries. The key idea behind the efficiency are our search region refinement and trip scheduling techniques for GGTS queries.
- We develop two heuristic solutions, TSH and SRH, to further reduce the processing overhead of GGTS queries. TSH does not consider all possible patterns of visiting POIs while

scheduling the trips for the group members, whereas SRH does not retrieve all candidate POIs from the database to ensure that the POIs that provide the minimum aggregate trip overhead distance are considered for scheduling trips.

- We run extensive experiments with real datasets to show the efficiency and effectiveness of our optimal and heuristic algorithms.

The remaining of the thesis is organized as follows. In Chapter 2, we formulate the problem of GGTS queries and in Chapter 3, we discuss the related work. Chapter 4 presents the system architecture. In Chapter 5, we propose the optimal approach and in Chapter 6, we discuss our heuristic solutions. Chapter 7 presents the experimental results and Chapter 8 concludes the thesis.

Chapter 2

Problem Formulation

We assume that the information of POIs are indexed using an R^*-tree [6] on the database of a location-based service provider(LSP). When a group sends a GGTS query to a LSP, the LSP evaluates the query using the POIs stored on its database and returns the answer to the group. In a GGTS query, a group of n members specify their sources, destinations, a set of m required POI types and for every POI type, the number of group members who need to visit each required POI type. The query returns n number of trips for n group members. A GGTS query for a group is formally defined as follows.

***Definition 1.*[Generalized Group Trip Scheduling (GGTS) Queries.]** Given a set of POIs $\mathbb{P}$, a set of n independent source locations $S = \{s_1, s_2, s_3, \ldots, s_n\}$ and their corresponding destination locations $D = \{d_1, d_2, d_3, \ldots, d_n\}$ of n number of members in a group $U = \{u_1, u_2, u_3, \ldots, u_n\}$, a set of m required POI types, $\mathbb{C} = \{c_1, c_2, c_3, \ldots, c_m\}$, a set of numbers of members who need to visit the corresponding required POI types, $N = \{n_1, n_2, n_3, \ldots, n_m\}$ for $1 \leq n_i \leq n$, a generalized group trip scheduling query (GGTS) returns a set of n trips, $T = \{T_1, T_2, T_3, \ldots, T_n\}$ for n members of the group such that the following constraints are satisfied:

- the trips minimizes aggregate trip overhead distance $AggTripOvDist$
- each trip visits any number of required POI types from 0 to m
- exactly n_i number of trips visit the same POI of type c_i

For any two point locations x_1 and x_2 in a 2-dimensional space, let the Function $Dist(x_1, x_2)$ return the distance between x_1 and x_2 in road networks. Let $p^k_{c_j}$ denote a POI of type $c_j \in \mathbb{C}$ with ID k. Then, for $T_i = \{p_1, p_2, p_3\}$ the trip distance $TripDist_i$ of u_i for T_i is computed as $Dist(s_i, p_1) + Dist(p_1, p_2) + Dist(p_2, p_3) + Dist(p_3, d_i)$, if the sequence of visiting POI types is specified by the group. On the other hand, if the sequence is flexible then $TripDist_i$

is measured as the minimum of the trip distances computed by considering all possible combinations of visiting p_1, p_2 and p_3. The trip overhead distance of a group member u_i is measured by deducting the distance between the source (s_i) and destination (d_i) locations from the trip distance $TripDist_i$. Thus, the trip overhead distance of a group member u_i is computed as $(TripDist_i - Dist(s_i, d_i))$ and the aggregate trip overhead distance $AggTripOvDist$ is measured as the summation of the trip overhead distances.

Table 2.1 summarizes the notations used in this thesis.

Table 2.1: Notations and their meanings

Notation	Symbol
c_j	A POI type
$p^k_{c_j}$	A POI of type c_j with ID k
u_i	A group member
n_i	A number of group members who visit the same POI of type i
T_i	The trip for group member u_i
$Dist(.,.)$	The distance between two points in road networks
$TripDist_i$	The trip distance of of a group member u_i
$AggTripOvDist$	An aggregate trip overhead distance

Chapter 3

Related Work

We discuss the related work in four sections. In Sections 3.1–3.5, we discuss the existing work on trip planning, group trip planning, group trip scheduling, and Traveling Salesman problems respectively, and differentiate GGTS queries from them.

3.1 Trip Planning Algorithms

Researchers have developed efficient algorithms for processing trip planning queries and variants [4, 5]. These algorithms assume that the trip is planned for a single user. In [7], the trip planning algorithm allows a user to specify multiple constraints like visiting a bank before watching a movie, filling up the gas before going to have dinner at a restaurant for planning trips. In [8], the work considers public pictures of past tourists', popular POIs, time available to the user etc while planning a trip for a user. In [9], the work focuses on protecting location privacy of users while planning their trips. Trip planning algorithms are not adaptable for GGTS queries, as it would require planning all possible trips for every member and then check which trip combination gives the total aggregate trip overhead distance. Thus, the application of trip planning algorithms for GGTS queries would incurs an extremely high processing overhead.

3.2 Group Trip Planning Algorithms

Though a group trip planning (GTP) query [1, 10–12] and its variants [13] allow more than one users to participate in the trip, still a single trip is planned for the group with the assumption that all group members together visit the required POI types. Researchers first addressed GTP queries in the Euclidean space [10]. The works proposed in [1, 10–12] solves the problem in road networks. In [13], the authors proposed an efficient solution for a dynamic GTP query, where a group member is allowed to join or leave the group trip anytime. A GGTS query is

different from a GTP query as it plans independent trip for every group member and allows any specified number of group members to visit the same POI of a required type.

3.3 Group Trip Scheduling Algorithms

A group trip scheduling (GTS) has been recently addressed in [2]. Though a GTS query plan independent trip for every group member with an aim to minimize the total aggregate trip distance, the query has restrictions in its applicability because it assumes that each POI type is visited by a single group member, which is not the case in reality. Furthermore, due to this assumption the solution proposed for the GTS query is also not adaptable for the GGTS query.

3.4 Traveling Salesman Problems

The Traveling Salesman Problems (TSPs) and variants have been extensively studied in the literature. For a given graph, the TSP problem [14] finds the shortest tour passing through all vertices of the graph. The Generalized Traveling Salesman Problem (GTSP) [15], the vertices are grouped into types and the GTSP finds the shortest tour visiting a single vertex from every group. In the Generalized Traveling Salesman Path Problems GTSPP [16], the salesmen have a fixed source and destination location and the types of vertices that a salesman visits are ordered. In the Trip Orienteering Problem (TOP) [17], the team members aim to maximize the total score obtained by visiting the vertices in the graph within a limited time period. However none of these problems are solved for large datasets like GGTS queries.

3.5 Other Queries

In [18], the authors have introduced query processing in spatial databases in road networks in stead of in Euclidean space. The work in [9] deals with the case where a user after starting from a source location wants to visit certain types of POIs bearing in mind that his trip distance is minimum and then departs for his destination without disclosing his source and destination locations to the service provider. The authors have tried to assess the problem using two approaches-providing false and cloaked locations. In case of providing false locations they have used elliptical properties while in case of cloaked locations they have combined triangular inequalities with elliptical properties to solve the cause. It is actually quite different from our problem in a sense that it deals with a single user and also privacy is its main concern while in our case we are dealing with a group and privacy is not an issue. In [19] the authors address the situation when a group of people want to visit a single POI type such that the aggregate trip

distance is minimized namely Group Nearest Neighbour(GNN) query. The difference of this problem with ours is that all the members visit a single POI and the aggregate trip distance is measured with respect to a single POI and the current location of the members. In [20] the authors have used a multi-step algorithm to process a GNN query. The work in [21] also deals with the same problem stated in [10] but proposes a redefined iterative approach for an optimal solution and introduces a new approach names Progressive Group Neighbour Exploration which assures that the solution works for both in Euclidean spaces and road networks. In [22] the authors have proposed a solution for the same query stated in [19] but the difference being the people of a group not wanting to disclose their locations to the group members or the service provider. Here,the members of the group provide their respective regions where they might be for keeping their locations private and the algorithm proposed by the authors deal with this situation by filtering distance intersection attack thus keeping the locations of the members private. Though the problem deals with a group of people in our problem there is no case of privacy of a member's location. In [23] the authors look into the problem stated in [19] but has taken account the impact of obstacles on travel distance. Using different pruning techniques the authors propose two algorithms for solving the problem efficiently. The main difference of this with our problem is this that in our problem we do not take into account the issue of obstacles.

Chapter 4

System Architecture

We use the same architecture that has been generally used for any group based location-based service. In Figure 4.1, we show an overview of the system architecture. A coordinator of a group sends a GGTS query request to the location service provider (LSP) and provides required parameters as mentioned in the definition of the GGTS query. All POI information is indexed using an R^*-tree [6] in the database. The LSP incrementally retrieves POIs from the database until it identifies the scheduled trips that minimize the aggregate trip overhead distance of the group. It then sends the trips of each member to the group coordinator who shares it with the other group members.

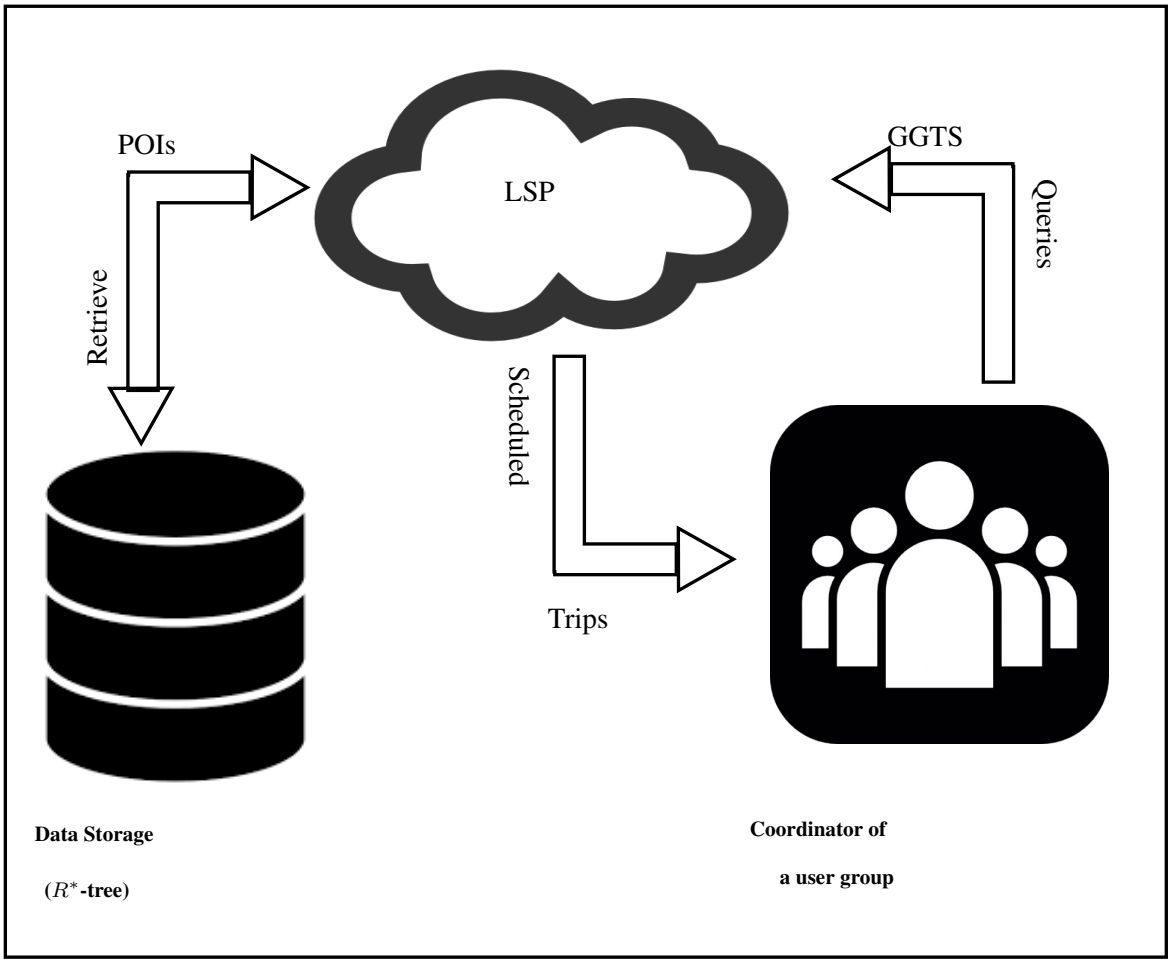


Figure 4.1: System Architecture

Chapter 5

An Optimal Approach

In this section, we present our approach to find the optimal answer for GGTS queries. The underlying idea of our approach is to incrementally retrieve POIs of required POI types from POI database until the optimal solution is found. After the retrieval of one or multiple POIs, our approach schedules the trips using the retrieved POIs and computes the upper bound of the aggregate trip overhead distance. The upper bound of the aggregate trip overhead distance is used to refine the POI search region. The retrieval of the POIs ends when all POIs within the refined search regions are retrieved.

The remaining part of this section is organized as follows. In Chapter 5.1 we discuss the concepts of known and search regions. Then we present the techniques for refining the search space and the terminating condition in Sections 5.2 and Chapter 5.3, and we propose an algorithm for finding the optimal answer in Chapter 5.4. Finally, we give an overview of our optimal approach for processing GGTS queries in Figure 5.4 using a flowchart.

5.1 Preliminaries

Similar to GTP [24], GTS [5] queries, we use the concept of known region and search region to refine our search space. However, the way existing work computes the search region becomes invalid for GGTS queries due to their restrictions for visiting POI types. Generally, all the POIs are indexed using an R^*-tree and stored in the POI database which are retrieved incrementally from the POI database. The known region represents the area which has already been explored that means all POIs inside the known region have been retrieved from the POI database. The search region represents the region that we need to explore for finding an optimal solution.For GGTS queries, we refine the POI search region using multiple ellipses. In Figure 5.1 the known region contains the POIs p_1^3 and p_4^3, that is those POIs have been retrieved from the POI database. The search region contains p_1^2, p_2^3 which indicates that we need to retrieve those POIs from the

POI database for finding optimal solution.

For computing the known region of a GGTS query, we incrementally retrieve the nearest POIs with respect to the geometric centroid G of the n source-destination pairs of the group members as stated in [2]. This approach uses the best-first search (BFS) technique to find the POIs of the required POI types that are indexed using an R^*-tree [6] in the database. If the BFS discover p_j as the first nearest POI with respect to G, the circular region centered at G with radius r equal to the Euclidean distance between G and p_j is the known region. After each retrieval of the next nearest POI, r is updated with the Euclidean distance from G to the last nearest POI retrieved from the database.

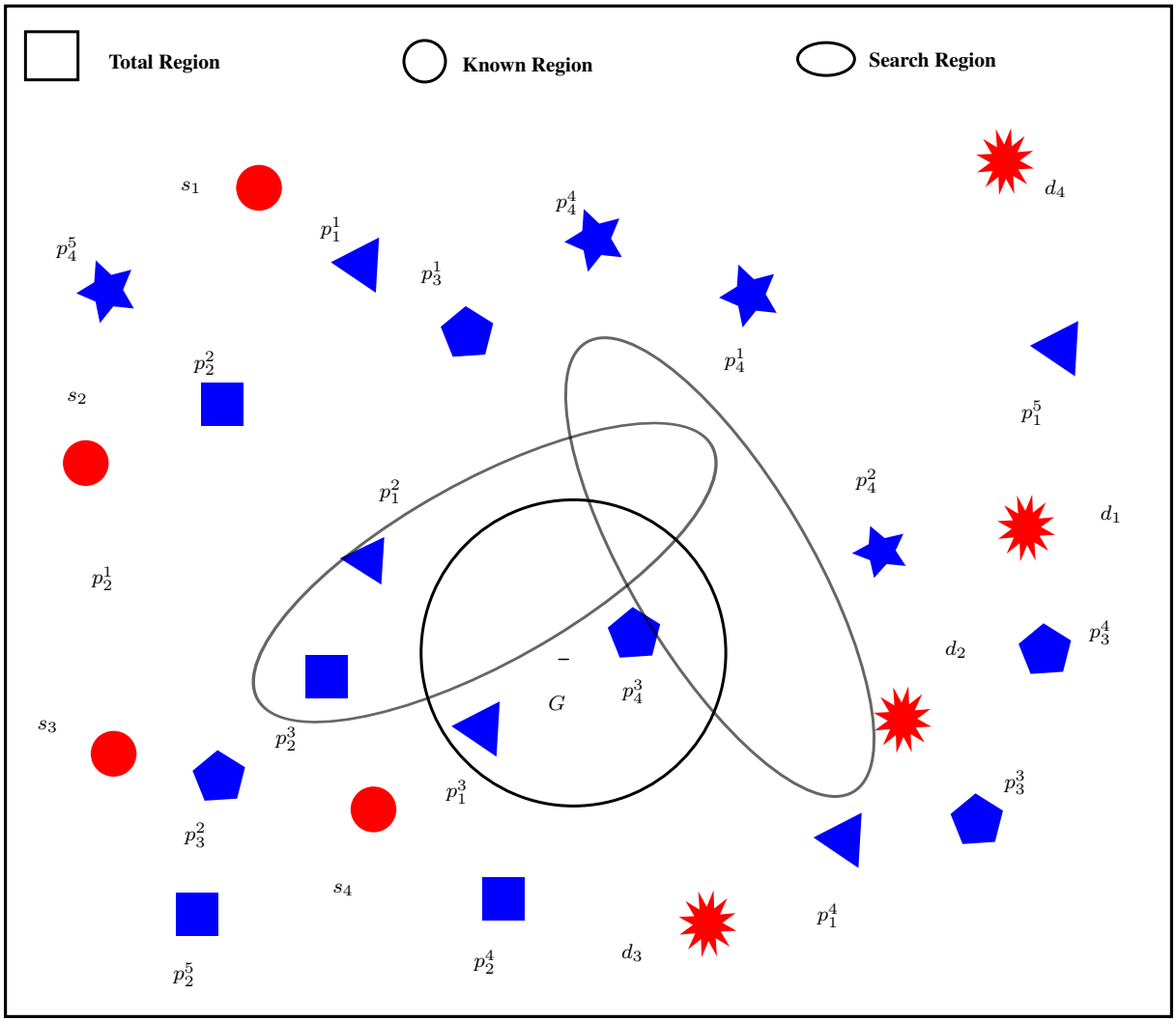


Figure 5.1: Known region and search Region

5.2 Refinement of the Search Region

We have used two search region refinement techniques- one for a particular POI type c_i and the other for n number of users. Any POI p outside the search region do not need to be picked as it cannot improve the $AggTripOvDist$ of the group. It should be noted that in literature

[2,4,5,7,24,25] different search region refinement techniques have been introduced for pruning POIs but they are not applicable for this query. We have basically used the theorem of triangular inequality and elliptical properties to prove our claims.

The notations that have been used are defined below:

- s_c: centroid of n source locations.

$$s_c = \frac{s_1 + s_2 + \ldots + s_n}{n}$$

- d_c: centroid of n destination locations.

$$d_c = \frac{d_1 + d_2 + \ldots + d_n}{n}$$

- $TripDist_i$: the current trip distance of a group member u_i among the scheduled trips.
- $AggTripDist$: the current minimum total trip distance of the group, $\sum_{i=1}^{n} TripDist_i$
- $AggTripOvDist$: the current minimum aggregate trip overhead distance of the group.

$$AggTripOvDist = \sum_{i=1}^{n} TripDist_i - Dist(s_i, d_i)$$

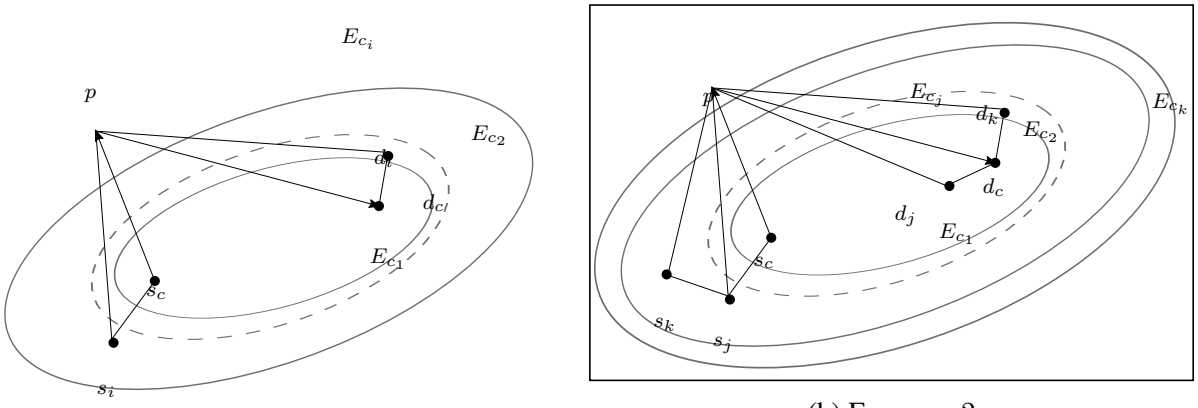
Theorem 1 shows the search region refinement technique for a particular POI type c_i.

Theorem 1. *The search region for a POI type c_i can be refined as an ellipse E_{c_i}, where the foci of E_{c_i} are at s_c and d_c, and the length of the major axis is equal to $\frac{AggTripOvDist}{n_i} + max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + max_{i=1}^{n} Dist(s_i, d_i)$.*

Proof. Suppose, a POI p of type c_i lies outside the ellipse E_{c_i} and exactly one of the group members is expected to visit that POI type. Assuming, when a group member u_i having source location s_i and destination location d_i visits that POI type, the group has the smallest aggregate trip overhead distance including that POI, we will be able to prune that POI p if the smallest aggregate trip overhead distance including p is greater than the current aggregate trip overhead distance that is $AggTripOvDist_p > AggTripOvDist$.

As p lies outside the ellipse E_{c_i} the sum of distances from the foci of the ellipse to p must be greater than the length of the major axis of the ellipse. Hence,

$$\begin{aligned} Dist(s_c, p) + Dist(d_c, p) > AggTripOvDist + \\ max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + \\ max_{i=1}^{n} Dist(s_i, d_i) \end{aligned} \tag{5.1}$$



(a) For $n_i = 1$

(b) For $n_i = 2$

Figure 5.2: Proof of Theorem 1 for search space refinement

From the theorem of triangular inequality,

$$Dist(s_i, s_c) + Dist(s_i, p) > Dist(s_c, p)$$
$$Dist(d_i, d_c) + Dist(d_i, p) > Dist(d_c, p)$$

which yields

$$\begin{aligned} Dist(s_i, s_c) + Dist(s_i, p) + Dist(d_i, d_c) + \\ Dist(d_i, p) > Dist(s_c, p) + Dist(d_c, p) \end{aligned} \tag{5.2}$$

Taking equation 5.1 and 5.2 into account,

$$\begin{aligned} Dist(s_i, s_c) + Dist(s_i, p) + Dist(d_i, d_c) + \\ Dist(d_i, p) > AggTripOvDist + \\ max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + \\ max_{i=1}^{n} Dist(s_i, d_i) \end{aligned}$$

Taking $Dist(s_i, s_c)$ and $Dist(d_i, d_c)$ to the right hand side of the inequality we find

$$\begin{aligned} Dist(s_i, p) + Dist(d_i, p) > AggTripOvDist \\ + max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + \\ max_{i=1}^{n} Dist(s_i, d_i) - Dist(s_i, s_c) - Dist(d_i, d_c) \end{aligned} \tag{5.3}$$

Since,

$$\begin{aligned} max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} - Dist(s_c, s_i) - \\ Dist(d_c, d_i) \geq 0 \end{aligned}$$

and

$$Dist(s_i, d_i) \leq max_{i=1}^{n}\{Dist(s_i, d_i)$$

Equation 5.3 yields,

$$Dist(s_i, p) + Dist(d_i, p) - Dist(s_i, d_i) > AggTripOvDist$$

For $n_i = 1$, we find $Dist(s_i, p) + Dist(d_i, p) - Dist(s_i, d_i)$ to be the aggregate trip overhead distance including POI p. So,

$$AggTripOvDist^p > AggTripOvDist$$

Now, there may also arrive cases when a certain POI type is expected to be visited by more than one group member. Let p be a POI of type c_i outside the ellipse E_{c_i} and two group members($n_i = 2$) are expected to visit that POI type. Suppose, two group members u_j and u_k having source and destination locations (s_j, d_j) and (s_k, d_k) respectively provide the smallest aggregate trip overhead distance when they visit p. So, proving that the smallest aggregate trip overhead distance including POI p of type c_i is greater than the current aggregate trip overhead distance should suffice for pruning the POI of that particular type.

From elliptical property it can be stated that,

$$Dist(s_c, p) + Dist(d_c, p) > \frac{AggTripOvDist}{2} + max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + max_{i=1}^{n} Dist(s_i, d_i)$$

which yields

$$2 * [Dist(s_c, p) + Dist(d_c, p)] > AggTripOvDist + 2 * [max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + max_{i=1}^{n} Dist(s_i, d_i)] \quad (5.4)$$

From the theorem of triangular inequality,

$$\begin{aligned} Dist(s_j, s_c) + Dist(s_j, p) > Dist(s_c, p) \\ Dist(s_k, s_c) + Dist(s_k, p) > Dist(s_c, p) \\ Dist(d_j, d_c) + Dist(d_j, p) > Dist(d_c, p) \\ Dist(d_k, d_c) + Dist(d_k, p) > Dist(d_c, p) \end{aligned}$$

Hence,

$$\begin{aligned} Dist(s_j, s_c) + Dist(s_j, p) + Dist(s_k, s_c) + Dist(s_k, p) + \\ Dist(d_j, d_c) + Dist(d_j, p) + Dist(d_k, d_c) + \\ Dist(d_k, p) > 2 * [Dist(s_c, p) + Dist(d_c, p)] \end{aligned} \tag{5.5}$$

Taking equation 5.4 and 5.5 into account,

$$\begin{aligned} Dist(s_j, s_c) + Dist(s_j, p) + Dist(s_k, s_c) + Dist(s_k, p) + \\ Dist(d_j, d_c) + Dist(d_j, p) + Dist(d_k, d_c) + Dist(d_k, p) > \\ AggTripOvDist + 2 * max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + \\ 2 * max_{i=1}^{n} Dist(s_i, d_i) \end{aligned}$$

Taking $Dist(s_j, s_c), Dist(s_k, s_c), Dist(d_j, d_c)$ and $Dist(d_k, d_c)$ to the right hand side of the inequality we find

$$\begin{aligned} Dist(s_j, p) + Dist(s_k, p) + Dist(d_j, p) + Dist(d_k, p) > \\ AggTripOvDist + 2 * max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} + \\ 2 * max_{i=1}^{n} Dist(s_i, d_i) - Dist(s_j, s_c) - \\ Dist(s_k, s_c) - Dist(d_j, d_c) - Dist(d_k, d_c) \end{aligned} \tag{5.6}$$

Since,

$$\begin{aligned} 2 * max_{i=1}^{n}\{Dist(s_c, s_i) + Dist(d_c, d_i)\} - Dist(s_c, s_j) - \\ Dist(s_c, s_k) - Dist(d_c, d_j) - Dist(d_c, d_k) \geq 0 \end{aligned}$$

and

$$Dist(s_i, d_i) \leq max_{i=1}^{n}\{Dist(s_i, d_i)$$

Equation 5.5 can be written as

$$Dist(s_j, p) + Dist(s_k, p) + Dist(d_j, p) + Dist(d_k, p) - Dist(s_j, d_j) - Dist(s_k, d_k) > AggTripOvDist$$

For $n_i = 2$, $Dist(s_j, p) + Dist(s_k, p) + Dist(d_j, p) + Dist(d_k, p) - Dist(s_j, d_j) - Dist(s_k, d_k)$is the smallest aggregate trip overhead distance where members u_j and u_k visit POI p. So,

$$AggTripOvDist^p > AggTripOvDist$$

Similarly, it can be shown that this holds true for $n_i > 2$.

Thus, any POI $p$of type c_i can be pruned if it is outside the ellipse E_{c_i}. □

For our user based search region refinement technique, we use the following theorem from [2].

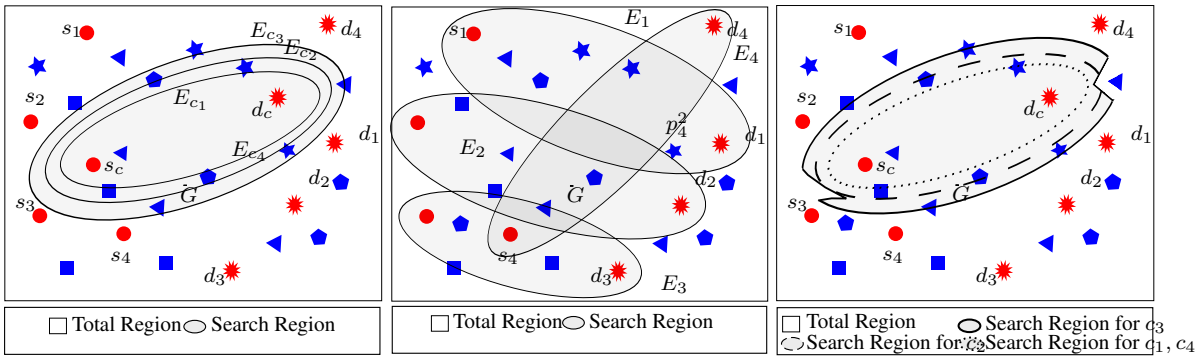
Theorem 2. *The search region can be refined as the union of n ellipses $E_1 \cup E_2 \cup \ldots \cup E_n$, where the foci of E_i are at s_i and d_i, and the major axis of E_i is equal to $AggTripOvDist + Dist(s_i, d_i)$.*

The following theorem combines Theorem 1 and Theorem 2 for pruning POIs for a particular POI type for a group of n users.

Theorem 3. *The search region for a POI type c_i can be refined as $E_{c_i} \cap (E_1 \cup E_2 \cup \ldots \cup E_n)$, where E_{c_i} represents the search region for a POI type c_i based on Theorem 1 and $(E_1 \cup E_2 \cup \ldots \cup E_n)$ represents the search region for a group of n users based on Theorem 2.*

Proof. According to Theorem 2 any POI of type c_i can be pruned outside the ellipse E_{c_i} as it cannot improve the $AggTripOvDist$ for the group. Then, according to Theorem 2 any POI outside $(E_1 \cup E_2 \cup ...E_n)$ can be pruned because it cannot improve the $AggTripOvDist$ of the group. Both theorems assert that any POI outside the region can be pruned. Hence, for a particular POI type c_i, any POI outside the region, $E_{c_i} \cap (E_1 \cup E_2 \cup \ldots \cup E_n)$, where E_{c_i} cannot be part of an optimal solution. □

Figure 5.3 gives an overview of the search region refinement techniques for the example scenario stated in Chapter 1. We find 4 ellipses $E_{c_1}, E_{c_2}, E_{c_3}, E_{c_4}$ as our type based search regions for POI types c_1, c_2, c_3, c_4 respectively. For user based search region we take the union of 4 ellipses E_1, E_2, E_3, E_4 for 4 users. Finally, combining both type and user based search regions for POI type c_1, c_2, c_3, c_4 our search region reduces as shown in Figure 5.3c.



(a) Type based search regions (b) User based search regions (c) Combined search regions

Figure 5.3: Search Region refinement techniques for example scenario in Chapter 1

5.3 Terminating Condition for POI retrieval

The optimal approach keeps on retrieving POIs incrementally until it finds an optimal answer. With each retrieval of a POI, the radius of known region increases and the aggregate trip overhead distance of the group is updated. Hence, the search region shrinks. When the known region covers the search region, no more minimization of aggregate trip overhead distance is possible. At this point, we stop retrieving POIs from the POI database and find our candidate POI set which contains the optimal solution.

5.4 Trip Scheduling

The optimal way of trip scheduling checks all possible combinations of users and POI types and finds out that particular user combination with POI types that together satisfies the constraints of a GGTS query and provides minimum aggregate trip overhead distance for the group. If $r_1, r_2, r_3, ..., r_n$ be the number of POIs of type $c_1, c_2, c_3, ..., c_n$ retrieved from database respectively, then after each retrieval of a POI we have to check $r_1 * r_2 * r_3 * ... * r_n$ number of combinations of POIs for finding out the optimal trips for n users.

Algorithm 1 shows the pseudo-code to evaluate scheduled trips for GGTS queries optimally. It takes the set of source and destination locations, S and D, respectively for a group of n members, the set of required m POI types $\mathbb{C}$, the number of members that who need to visit the required POI types $\mathbb{N}$ as input. The output is a set of n scheduled trips $T = \{T_1, T_2, \ldots, T_n\}$, where n trips together visit all POI types in $\mathbb{C}$ and one POI type is visited by exactly one or more than one trip as per the required input.

Function $Initialize()$ initializes G, the geometric centroid of source and destination locations of n users to the center of the known region. It declares n elliptic regions for n users as $ellipregions = \{E_1, E_2, \ldots, E_n\}$, where the foci of each ellipse E_i is initialized to the

Algorithm 1 $Optimal\_GGTS\_Approach(S, D, \mathbb{C}, f, qType)$

input : $S, D, \mathbb{C}, f, \mathbb{N}, A$
output: A set of trips, T

$Initialize()$;
$InitCombinationTable(\mathbb{C}, |\mathbb{C}|, |S|)$;
$InitIndexTable(\mathbb{C}, |\mathbb{C}|, |S|, \mathcal{P})$;
$Enqueue(Q_p, root, MinD(G, root))$;
while Q_p *is not empty* **do**
  if $end = 1$ **then**
    break;
  $\{p, d_{min}(p)\} \leftarrow Dequeue(Q_p)$;
  $r \leftarrow d_{min}(p)$;
  if p *is not a POI* **then**
    foreach *child node* p_c *of* p **do**
      $Enqueue(Q_p, p_c, MinD(G, p_c))$;
  else if $\tau(p) \in \mathbb{C}$ *and* $p \in E_{c_i} \cap (\cup_{i=1}^{n} E_i)$ **then**
    $P \leftarrow InsertPOI(p)$;
    if $init = 0$ *and* $CheckInclude(\mathbb{P}, \mathbb{C})$ **then**
      $ComputeTrip(S, D, \mathbb{C}, \mathbb{P}, \mathcal{P}, \mathcal{I})$;
      $init \leftarrow 1$;
      $isup \leftarrow true$;
    else if $init = 1$ **then**
      $isup \leftarrow UpdateTrip(\tau(p), S, D, \mathbb{C}, p, \mathcal{P}, \mathcal{I})$;
  if $isup = true$ *and* $init = 1$ **then**
    $ellipregions \leftarrow UpEllipticRegions()$;
  if $IsInCircle(G, r, ellipregions)$ **then**
    $end \leftarrow 1$;
return T

Table 5.1: Combination table for optimal GGTS approach

Id	Combination
1	$u_1 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
2	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_4$
3	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_3 \rightarrow c_4$
4	$u_1 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
5	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_3 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
6	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_4$
	$u_13 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
7	$u_1 : c_1 \rightarrow c_3 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
8	$u_1 : c_1 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
9	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$

source and destination location of a user and the length of the major axis is set to ∞. It further declares m elliptic regions for m number of POI types as $ellipregionstypebased = \{E_{c_1}, E_{c_2}, \ldots, E_{c_m}\}$, where the foci of each ellipse E_{c_i} is initialized to the centroid of source and destination locations of n users and the length of the major axis is set to ∞. Apart from declaring known region and search region this function initializes a priority queue Q_p to $\emptyset$ and other variables as follows: $r = 0$, $end = 0$, $isup = false$, and $init = 0$. The variable r represents the radius of current known region. Flags end and $isup$ indicate whether the terminating condition is true and trip distance of the group has been updated respectively. Variable $init$ is used to keep track whether initial trips have been computed. Then a *combination table* is constructed which contains all possible combinations of users who visit all possible combinations of the required POI types that satisfies all the constraints of a GGTS query. Each combination consists of n rows for n users and the columns represent the POI types a user has to visit to satisfy the conditions of the query. For a n member group, if $\{n_1, n_2, \ldots, n_m\}$ represents the number of members that are expected to visit required POI types $\{c_1, c_2, \ldots, c_m\}$ respectively, the number of rows for the combination table equals to ${}^nC_{n_1} * {}^nC_{n_2} * \ldots * {}^nC_{n_m}$.

Now that the combination table is constructed, we construct an *index table* that contains unique sequences of POI types a user has to visit from combination table. Each row contains a unique sequence of POI types a user has to visit and is assigned an index number. It also stores the index of that specific combination from combination table which contains this unique sequence of POI types. If a sequence is present in a combination multiple times the corresponding index of the combination is stored multiple times. For a larger group size or larger number of POI types the number of combinations increase rapidly as a result memory overflow may occur. To avoid this situation we introduce the concept of index table.

The algorithm starts searching from the $root$ of the R^*-tree and inserts the $root$ with $MinD(G, root)$ into a priority queue Q_p. Q_p stores its elements in order of their minimum distances from G, $d_{min}(p)$ that are determined by Function $MinD(G, p)$. For both Euclidean space and road networks, $MinD(G, p)$ returns the minimum Euclidean distance between G and p, where p represents a POI or a minimum bounding rectangle of a R^*-tree node. After that the algorithm removes an element p along with $d_{min}(p)$ from Q_p. At this step, the algorithm updates r, the radius of current known region. If p represents a R^*-tree node, then algorithm retrieves its child nodes and en-queues them into Q_p. On the other hand, if p is a POI of type c_i specified in $\mathbb{C}$ and falls inside , then it is added to candidate POI set P. The algorithm uses function $\tau(p)$ to determine the POI type of a POI p.

Function $CheckInclude(\mathbb{P}, \mathbb{C})$ checks whether the POI set $\mathbb{P}$ contains at least one POI from each required POI type in $\mathbb{C}$. When the initial POI set has been found for a GGTS query, Function $ComputeTripInitial(S, D, \mathbb{C}, P, \mathcal{P}, \mathcal{I})$ computes all possible trips for users taking into account all the combinations present in the combination table and finds an upper bound for the aggregate trip overhead distance of the group.

After finding an upper bound, if the algorithm retrieves any new POI p, it uses Function $UpdateTrip(\tau(p), S, D, \mathbb{C}, p, \mathcal{P}, \mathcal{I})$ where it checks if the inclusion of POIp improves the aggregate trip overhead distance by considering all the combinations present in combination table and the candidate POI set and updates $isup$ accordingly.

Function $UpEllipticRegions$, the algorithm updates the elliptic bound for all n users for m POI types, where for a POI type c_i the elliptic search region is $E_{c_i} \cap \{E_i, E_2, \ldots, E_n\}$. The bounds for the elliptic search regions are determined using both Theorem 1 and 2.

The algorithm checks the terminating condition of our GGTS queries using Function $IsInCircle(G, r, ellipreg$ which checks whether all m elliptic search regions is included by the current circular known region or not. If the terminating condition is true, the algorithm updates the terminating flag end to 1. At the end of the algorithm, it returns a set of scheduled trips T for n users that provide the minimum aggregate trip overhead distance.

Table 5.2: Index table for example scenario

Index	Combination	Index of combination table
1	$c_1 \rightarrow c_4$	1, 4, 6, 8, 9
2	$c_1 \rightarrow c_2 \rightarrow c_4$	1, 2,3, 3, 4, 5, 5, 6, 7, 7, 8, 9
3	$c_1 \rightarrow c_3 \rightarrow c_4$	3, 5, 7,
4	$c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$	1, 2, 4, 6, 8, 9

We elaborate our optimal trip scheduling technique with an example. Suppose, a group of 3 members, $\{u_1, u_2, u_3\}$ visit 4 POI types such that $3, 2, 1$, and 3 members visit POI types c_1, c_2, c_3 and c_4 respectively. Our first task is to construct the combination and index table accordingly. The combination table will contain ${}^3C_3 * {}^3C_2 * {}^3C_1 * {}^3C_3 = 9$ rows where each row can be

a possible solution and the index table is constructed from the combination table containing 4 unique sequences of POI types.

For an initial POI set, $\mathbb{P} = p_1^1, p_2^1, p_3^1, p_3^2, p_4^1$ we have to check all the 9 combinations in Table 5.1 with the candidate POI set. We explain this scenario by taking into account combination number 1 from the combination table. For combination number 1 the aggregate trip overhead distance of the group is calculated as follows.

Table 5.3: Calculation of trip distances for combination 1 with POIs: $p_1^1, p_2^1, p_3^1, p_4^1$

$s_1 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_3^1 \rightarrow p_4^1$	120.6	
$s_2 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_4^1$	80.5	294.8
$s_3 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_4^1$	93.7	

Table 5.4: Calculation of trip distances for combination 1 with POIs: $p_1^1, p_2^1, p_3^2, p_4^1$

$s_1 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_3^2 \rightarrow p_4^1$	140.8	
$s_2 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_4^1$	80.5	315
$s_3 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_4^1$	93.7	

Table 5.5: Calculation of trip distances for combination 1 with POIs: $p_1^1, p_2^2, p_3^1, p_4^1$ after retrieval of POI p_2^2

$s_1 \rightarrow p_1^1 \rightarrow p_2^2 \rightarrow p_3^1 \rightarrow p_4^1$	89.7	
$s_2 \rightarrow p_1^1 \rightarrow p_2^2 \rightarrow p_4^1$	86.5	252.4
$s_3 \rightarrow p_1^1 \rightarrow p_2^2 \rightarrow p_4^1$	76.2	

With the initial POI set our solution will either be $\{p_1^1, p_2^1, p_3^1, p_4^1\}$ or $\{p_1^1, p_2^1, p_3^2, p_4^1\}$. For combination number 1 and POI set $p_1^1, p_2^1, p_3^1, p_4^1$ we find the aggregate trip overhead distance from Table 5.3 and for POI set $p_1^1, p_2^1, p_3^2, p_4^1$ the distances are calculated in Table 5.4. We take the minimum of the two distances 294.8 and by deducting $Dist(s_1, d_1) + Dist(s_2, d_2) + Dist(s_3, d_3)$ store it for combination 1.

The same procedure is followed for calculating aggregate trip overhead distances for other combinations in combination table. Finally, the combination with the least aggregate trip overhead distance gives an upper bound for our aggregate trip overhead distance of the group.

After calculating the upper bound, say a new POI p_2^2 is retrieved from the database. Thus our candidate POI set now becomes $\mathbb{P} = p_1^1, p_2^1, p_2^2, p_3^1, p_3^2, p_4^1$ and our solution can be $\{p_1^1, p_2^1, p_3^1, p_4^1\}$ or $\{p_1^1, p_2^2, p_3^1, p_4^1\}$ or $\{p_1^1, p_2^1, p_3^2, p_4^1\}$ or $\{p_1^1, p_2^2, p_3^2, p_4^1\}$. We have checked for the POI set $\{p_1^1, p_2^1, p_3^1, p_4^1\}$ and $\{p_1^1, p_2^1, p_3^2, p_4^1\}$ in the earlier step. Now we shall check for POI set $\{p_1^1, p_2^2, p_3^1, p_4^1\}$ or $\{p_1^1, p_2^2, p_3^2, p_4^1\}$ in a similar way as explained above for each combination in the combination table to see if the newly retrieved POI can improve the aggregate trip overhead distance.

After retrieval of POI p_2^2 we find new trip distances for combination 1 for POI set including p_2^2 and see that $p_1^1, p_2^2, p_3^1, p_4^1$ improves our previous aggregate trip overhead distances. Similarly, we calculate for all other combinations and say $p_1^1, p_2^2, p_3^1, p_4^1$ gives the least trip distance for with

Table 5.6: Calculation of trip distances for combination 1 with POIs: $p_1^1, p_2^2, p_3^2, p_4^1$ after retrieval of POI p_2^2

$s_1 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_3^2 \rightarrow p_4^1$	160.2	
$s_2 \rightarrow p_1^1 \rightarrow p_2^1 \rightarrow p_4^1$	86.5	340.4
$s_3 \rightarrow p_1^1 \rightarrow p_2^2 \rightarrow p_4^1$	93.7	

combination 1. Hence, we shall update the aggregate trip overhead distance of the group which also updates the search region in our approach.

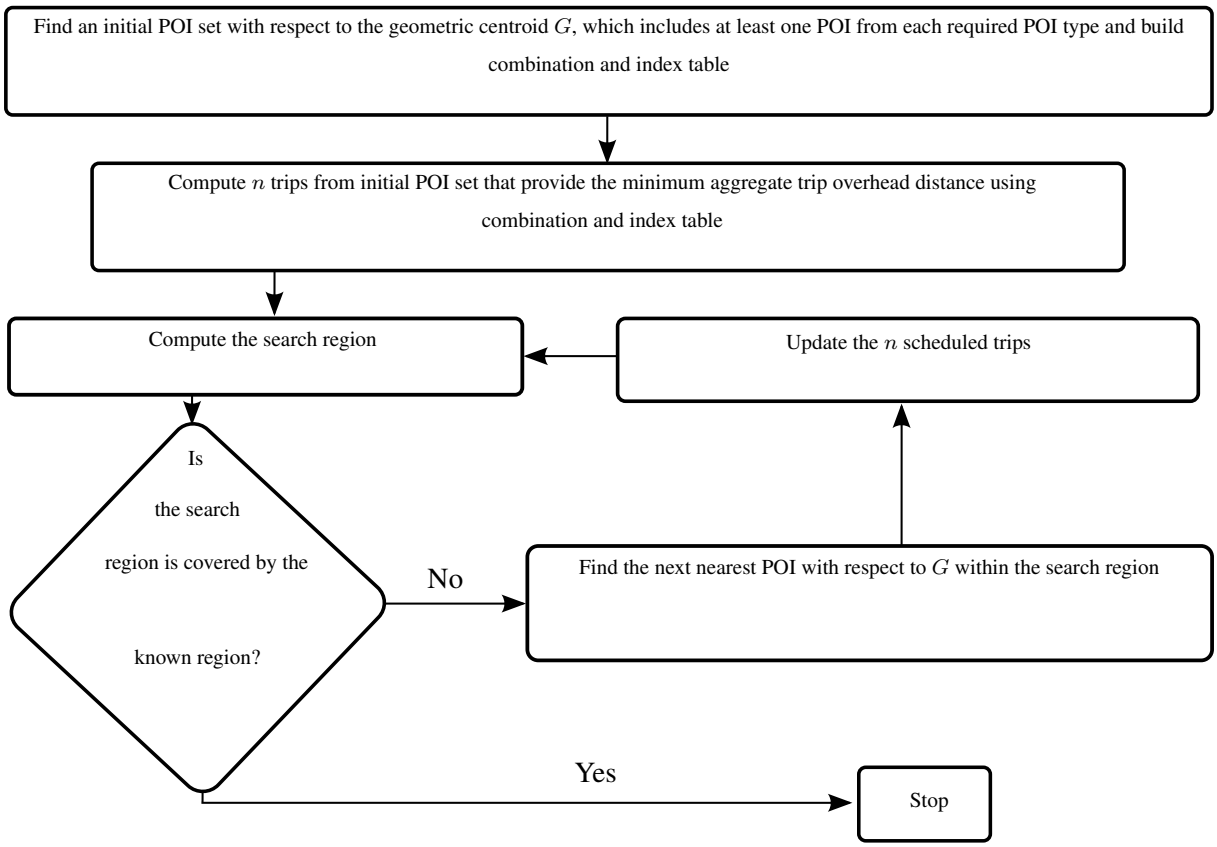


Figure 5.4: An overview for processing optimal GGTS query

Chapter 6

Heuristic Approach

Since, the problem of finding an optimal answer for a given GGTS query is a NP-Hard problem, the processing overhead is quite high for a large group size, a large number of POI types or large POI dataset. Thus, when the priority is to find the query answer in real time we can use heuristic approaches instead of the optimal one to evaluate the GGTS queries. The aim of the heuristic approaches is to reduce the processing time in return of sacrificing the accuracy of the answer slightly.

The processing time for our optimal solution comes from (ii) the refinement of the POI search space and retrieving the POIs, and (ii) scheduling the trips. We propose two heuristic solutions, : trip scheduling heuristic (TSH) and search space refinement heuristic (SRH), which reduces the processing overhead considerably for larger group size, number of POI types or dataset but does not guarantee the optimality of solution. TSH aims to reduce the required time for scheduling the trips whereas SRH aims to reduce the time for the refinement of the POI search space and retrieving the POIs. In the following two sections, we elaborate how TSH and SRH evaluates the GGTS queries. Then in Chapter 6.3, we compare the heuristic approaches with the optimal one.

6.1 Trip Scheduling Heuristic (TSH)

$TSH - GGTS$ evaluates a GGTS query quite similarly to the process described in Chapter 5. The only difference is while scheduling of trips in stead of checking all the combinations of users and POI types we curtain our checkings to a limited number of combinations. TSH uses the same search region refinement techniques proposed in Section 5.2 for finding candidate POIs of required POI types which contains the optimal answer. For trip scheduling, when TSH finds at least one POI from each required POI type and calculates the aggregate trip overhead distance for each combination stored in combination table. Then TSH keeps only best t combinations for

which the aggregate trip distance is the minimum in the combination table and delete all other combinations. Next, when a POI p is retrieved from POI database TSH only checks with the existent t combinations in the combination table for updating the trip distances for the group. The choice of t is a tricky matter, as larger t increases processing overhead whereas a smaller t reduces the accuracy of the answer. We have used $t = 10$ while running the experiments.

For the same example scenario described in Section 5.4, for initial POI set $p_1^1, p_2^1, p_3^1, p_3^1, p_4^1$ we calculate the trip distances for each combination similarly and for $t = 5$ store only the best five combination that gives us minimum aggregate trip overhead distance. Say, combination number $1, 3, 5, 7, 9$ gives us the the minimum aggregate trip overhead distance. Thus we only keep this combinations in the combination table and discard all other combinations. Hence, after a POI retrieval we shall now check only with these 5 combinations rather than checking with 9 combinations as we did in Section 5.4.

Table 6.1: Combination table for TSH-GGTS approach

Id	Combination
1	$u_1 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
3	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_3 \rightarrow c_4$
5	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_3 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
7	$u_1 : c_1 \rightarrow c_3 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_4$
9	$u_1 : c_1 \rightarrow c_2 \rightarrow c_4$
	$u_2 : c_1 \rightarrow c_4$
	$u_3 : c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$

6.2 Search Region Refinement Heuristic (SRH)

$SRH - GGTS$ uses a different approach for retrieving POIs of required types from the POI database. It does not use the concept of known region and search region used previously in Chapter 5 and Section 6.1. For finding out the candidate POI set of required types $SRH - GGTS$ keeps on retrieving the nearest POIs with respect to source location s_i and destination location d_i for a user u_i until it retrieves at least one POI from each required POI type. Taking the union of all these POIs retrieved for n users SRH finds the candidate POI set. Then for scheduling of the trips SRH checks the candidate POIs with all the combinations shown in Table 5.1.

For the same example scenario described in Section 5.4, say our candidate POI set, $\mathbb{P} = \{p_1^1, p_1^2, p_1^3, p_2^1, p_2^2, p_3^1, p_3^2, p_4^1, p_4^2, p_4^3, p_4^4\}$. We shall not further retrieve any POIs from the POI database. With this candidate POI set, we check with 9 combinations in Table 5.1.

6.3 Comparative Analysis

Table 6.2 shows a comparative analysis in terms of the features of the optimal and heuristic approaches. Both optimal approach and TSH use the concepts of the known region and the search region, whereas SRH greedily retrieves the POIs and does not use the concepts of the known region and the search region. The retrieved candidate POIs from the database for both the optimal approach and TSH guarantees that the candidate POI set includes the POIs that provide the minimum aggregate trip overhead distance. N

SRH uses the same trip scheduling algorithm as the optimal one, where the trip scheduling algorithm finds the trips with the minimum aggregate trip overhead distance using the candidate POIs. On the other hand, though the candidate POI set includes the optimal answer, the trip scheduling algorithm for TSH does not check all possible combinations of POIs while computing the trips using the candidate POIs.

Though for both optimal and TSH, the candidate POI set includes the optimal answer, the retrieved number of candidate POIs for SRH is greater than or equal to that of the optimal approach. This is because the POI search region is refined using the upper bound of the aggregate trip overhead distance and since TSH considers a small number of combinations for scheduling trips, the upper bound of the aggregate trip overhead distance might be greater than that of the optimal approach.

Table 6.2: Overview of O-GGTS, TSH-GGTS and SRH-GGTS for processing GGTS queries

O-GGTS	**TSH-GGTS**	**SRH-GGTS**
uses the concept of known region and search region	uses the concept of known region and search region	does not use the concept
Candidate POI sets contains optimal solution	Candidate POI sets contains optimal solution	may or may not contain optimal solution
Trip Scheduling checks all possible combinations	Trip Scheduling checks best t combinations	checks all possible combinations

Chapter 7

Experiments

In this section, the performance of the optimal and two heuristic approaches for processing GGTS queries in road networks are evaluated based on a wide range of extensive experiments by varying different parameters. We propose the first solutions for GGTS queries and existing work on GTP and GTS queries are not adaptable for processing GGTS queries (please see Chapter 3). Hence we cannot use any existing work to compare with our proposed solutions. We perform a comparative analysis among our optimal (O-GGTS) and heuristic approaches (TSH-GGTH and SRH-GGTS).

We used California [26] data set that contains 87635 POIs of 63 different types. The road network of California has 21048 nodes and 21693 edges. The whole data space is normalized to 1000x1000 sq. units. An R^*-tree is used to store all the POIs of a data set and a in-memory graph data structure is used to store the road network.

We used an Intel Core i5 machine with 3.30 GHz CPU and 8GB RAM to run the experiments. For each set of experiment, we measured the average processing time and I/O Overhead of different approaches. We also calculate the accuracy of the query answers given by two heuristic approaches. For a GGTS query sample, we measure the accuracy of the query answer for a heuristic (TSH or SRH) by comparing $AggTripOvDist$ of TSH or SRH with $AggTripOvDist$ of O-GGTS. We ran 100 independent GGTS queries having random source and destination locations, and then took the average of processing time, I/O overhead and accuracy.

We performed several sets of experiments by varying the following parameters: (i) the group size n, (ii) the number of specified POI types m, (iii) the query area A, i.e., the minimum bounding rectangle covering the source and destination locations. Table 7.1 shows the range and default values used for each parameter. To observe the effect of a parameter in an experiment, the value of the parameter is varied within its range, and other parameters are set to their default values.

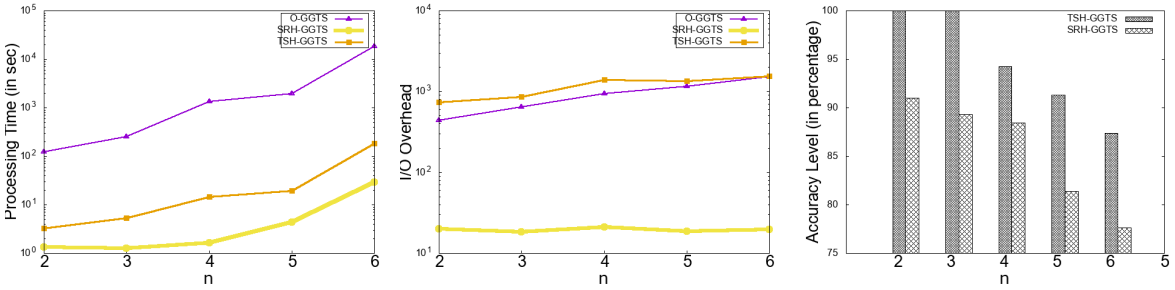
Experimental results for processing GGTS queries in road networks using optimal and two

Table 7.1: Parameter settings for GGTS queries

Parameter	Values	Default
Group size(n)	2, 3, 4, 5, 6	3
Number of POI types (m)	1,2, 3, 4 , 5 , 6	3
Query area(A) (in sq. units)	50x50, 100x100, 150x150, 200x200, 250x250, 300x300	100x100

heuristic approaches are shown below varying various parameters. Also, the accuracy of THS-GGTS and SRH-GGTS is measured comparing with optimal approach.

Effect of Group Size (n): The query processing time increases with the increase of group size n for all three approaches. This is because the number of combinations of the group members increase with the increase of group size which increases the road network distance computations. THS-GGTS has a constant I/O overhead for varying n whereas for both O-GGTS and TSH-GGTS the I/O overhead increases with increasing n. This is because SRH-GGTS retrieves almost a constant amount of POIs for consideration whereas for the other two approaches the number of POIs retrieved varies with increasing group size. Though the query processing time and I/O overhead for SRH-GGTS approach is the least of all three, it has a lower level of accuracy. TSH-GGTS ensures a higher level of accuracy with a higher query processing time and I/O overhead.

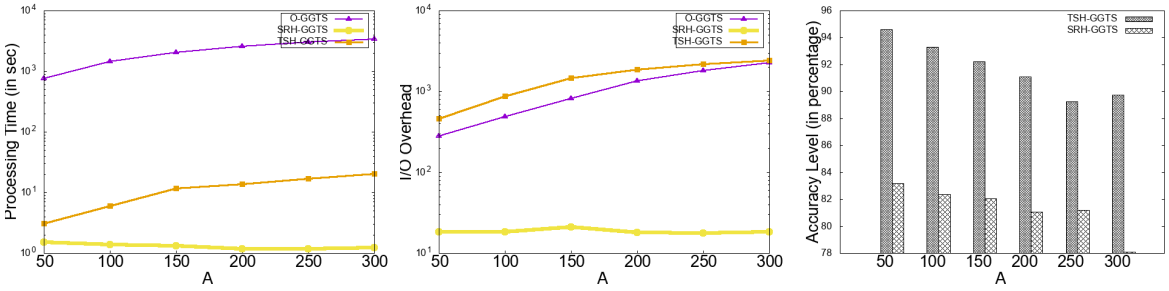


(a) Processing time (b) I/O Overhead (c) Accuracy level

Figure 7.1: Effect of n

Effect of Query Area (A): Figure 7.2(a) and (b) shows that query processing time increases with the increase of A while the I/O overhead almost remains constant for all three approaches. Like before, the TSH-GGTS approach ensures higher accuracy over SRH-GGTS but has a higher query processing time and I/O overhead than SRH-GGTS.

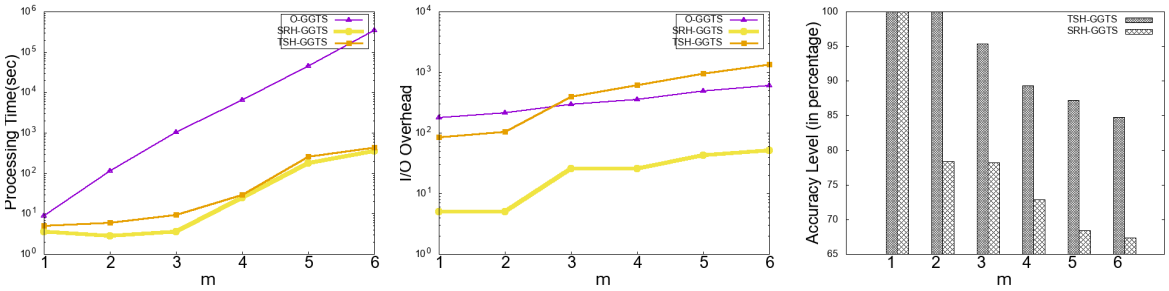
Effect of Number of POI Types (m): As the number of require POI types a group needs to visit increases the query processing overhead increases because of the increasing POI combina-



(a) Processing time (b) I/O Overhead (c) Accuracy level

Figure 7.2: Effect of A

tions a user is required to visit increases. Furthermore, the query processing time for larger m remains almost same for SRH-GGTS and TSH-GGTS because in TSH-GGTS we curtain the possible combinations of users but the combinations of the required POI types remain same. The I/O overhead increases for a larger m for SRH-GGTS unlike before because of the increasing POI combinations that needs to be checked again. As usual SRH-GGTS has a lower I/O overhead than both optimal-GGTS and TSH-GGTS but the accuracy level for SRH-GGTS is lower compared to TSH-GGTS.



(a) Processing time (b) I/O Overhead (c) Accuracy level

Figure 7.3: Effect of m

Chapter 8

Conclusion

In this thesis, we introduced a novel query type, a GGTS query that enables friends and families to manage daily activities with convenience and optimized manner. We developed the first comprehensive solution for GGTS queries. Specifically, we proposed an optimal approach and two heuristic solutions TSH and SRH for efficient processing of GGTS queries in road networks. Our experiments with real datasets show that our optimal approach can find the exact query answers in real time when the group size is reasonable (e.g., upto 5 members). On the other hand, the heuristic solutions are preferable when the processing overhead increases for the optimal approach for large parameter settings (e.g., group size greater than 5). Experiments show that TSH requires on average 23299.64 times less the processing time than the optimal approach and can evaluate the query answer with a very high accuracy (on average 92.95%). On the other hand, SRH requires on average 23330.07 times less the processing time than the optimal approach and can evaluate the query answer with a reasonable accuracy (on average 77.26%).

References

[1] T. Hashem, S. Barua, M. E. Ali, L. Kulik, and E. Tanin, "Efficient computation of trips with friends and families," in *CIKM*, pp. 931–940, ACM, 2015.

[2] R. Jahan, T. Hashem, and S. Barua, "Group trip scheduling (GTS) queries in spatial databases," in *EDBT*, pp. 390–401, OpenProceedings.org, 2017.

[3] H. Li, H. Lu, B. Huang, and Z. Huang, "Two ellipse-based pruning methods for group nearest neighbor queries," in *International Workshop on GIS*, pp. 192–199, 2005.

[4] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *VLDB J.*, vol. 17, no. 4, pp. 765–787, 2008.

[5] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng, "On trip planning queries in spatial databases," in *SSTD*, pp. 273–290, 2005.

[6] http://rtreeportal.org/.

[7] H. Chen, W. Ku, M. Sun, and R. Zimmermann, "The multi-rule partial sequenced route query," in *SIGSPATIAL*, pp. 10:1–10, 2008.

[8] I. R. Brilhante, J. A. F. de Macêdo, F. M. Nardini, R. Perego, and C. Renso, "Where shall we go today?: planning touristic tours with tripbuilder," in *CIKM*, pp. 757–762, 2013.

[9] S. C. Soma, T. Hashem, M. A. Cheema, and S. Samrose, "Trip planning queries with location privacy in spatial databases," *World Wide Web*, vol. 20, no. 2, pp. 205–236, 2017.

[10] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik, "Group trip planning queries in spatial databases," in *SSTD*, vol. 8098 of *Lecture Notes in Computer Science*, pp. 259–276, Springer, 2013.

[11] S. Samrose, T. Hashem, S. Barua, M. E. Ali, M. H. Uddin, and M. I. Mahmud, "Efficient computation of group optimal sequenced routes in road networks," in *MDM*, pp. 122–127, 2015.

[12] E. Ahmadi and M. A. Nascimento, "A mixed breadth-depth first search strategy for sequenced group trip planning queries," in *MDM*, pp. 24–33, 2015.

[13] A. Tabassum, S. Barua, T. Hashem, and T. Chowdhury, "Dynamic group trip planning queries in spatial databases," in *SSDBM*, pp. 38:1–38:6, ACM, 2017.

[14] G. Laporte, "A concise guide to the traveling salesman problem," *JORS*, vol. 61, no. 1, pp. 35–40, 2010.

[15] B. Bontoux, C. Artigues, and D. Feillet, "A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem," *Computers & OR*, vol. 37, no. 11, pp. 1844–1852, 2010.

[16] M. N. Rice and V. J. Tsotras, "Engineering generalized shortest path queries," in *ICDE*, pp. 949–960, 2013.

[17] "The team orienteering problem," *European Journal of Operational Research*, vol. 88, pp. 464–474, 1996.

[18] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, pp. 802–813, 2003.

[19] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*, pp. 301–312, IEEE Computer Society, 2004.

[20] T. Seidl and H. Kriegel, "Optimal multi-step k-nearest neighbor search," in *SIGMOD*, pp. 154–165, 1998.

[21] E. Ahmadi and M. A. Nascimento, "A mixed breadth-depth first search strategy for sequenced group trip planning queries," in *MDM (1)*, pp. 24–33, IEEE Computer Society, 2015.

[22] T. Hashem, L. Kulik, and R. Zhang, "Privacy preserving group nearest neighbor queries," in *EDBT*, vol. 426, pp. 489–500, ACM, 2010.

[23] N. Sultana, T. Hashem, and L. Kulik, "Group nearest neighbor queries in the presence of obstacles," in *SIGSPATIAL/GIS*, pp. 481–484, ACM, 2014.

[24] T. Hashem, S. Barua, M. E. Ali, L. Kulik, and E. Tanin, "Efficient computation of trips with friends and families," in *CIKM*, pp. 931–940, 2015.

[25] H. Li, H. Lu, B. Huang, and Z. Huang, "Two ellipse-based pruning methods for group nearest neighbor queries," in *GIS*, pp. 192–199, 2005.

[26] "California road network data." https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm.

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Friday 19th October, 2018 at 7:20pm.