Trip Planning Queries for Subgroups in Spatial Databases

Tanzima Hashem[1], Tahrima Hashem[2], Mohammed Eunus Ali[1], Lars Kulik[3], and Egemen Tanin[3]

[1] Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Bangladesh,
{tanzimahashem, eunus}@cse.buet.ac.bd
[2] Department of Computer Science and Engineering, Dhaka University, Bangladesh,
tahrimacsedu14@gmail.com
[3] Department of Computing and Information System, University of Melbourne, Australia,
{lkulik, etanin} @unimelb.edu.au

Abstract. In this paper, we introduce a novel type of trip planning queries, a subgroup trip planning (SGTP) query that allows a group to identify the subgroup and the points of interests (POIs) from each required type (e.g., restaurant, shopping center, movie theater) that have the minimum aggregate trip distance for any subgroup size. The trip distance of a user starts at the user's source location and ends at the user's destination via the POIs. The computation of POI set for all possible subgroups with the straightforward application of group trip planning (GTP) algorithms would be prohibitively expensive. We propose an algorithm to compute answers for different subgroup size concurrently with less query processing overhead. We focus on both minimizing the total and maximum trip distance of the subgroup. We show the efficiency of our algorithms in experiments using both real and synthetic datasets.

1 Introduction

Location-based services that involve more than one user have become common in recent years [11, 13, 16]. A group of users located at different places may want to visit a number of points of interests (POIs) such as a restaurant, a shopping center, or a movie theater together, and then go towards their own individual destinations. Group trip planning (GTP) queries [6, 7, 15] return points of interests (POIs) of required types (e.g., restaurant, shopping center, movie theater) that minimize the aggregate trip distance with respect to the source and destination locations of the group members. The trip distance of a user starts at the user's source location and ends at the user's destination via the returned POIs and the aggregate trip distance is computed as the total or the maximum trip distance of group members.

Sometimes it may happen that the majority of the users in the group are located nearby, whereas a few users may be located far away from them. These distant users, considered as *outliers*, may change the query answer for the group and increase the trip distances of the other group members. In such scenarios, if the aggregate trip distance of a subgroup improves significantly compared to a complete group, the group may prefer

to select the POIs for the trip that minimize the aggregate trip distance for a subgroup. Other group members who are not included in the subgroup can join the trip in return of traveling a little bit more for the overall interest of the group. In Figure 1, for the entire group (of size 4), the pair of POIs (p_1'', p_2'') minimizes the maximum travel distance, while (p_1'', p_2') minimizes the maximum travel distance if we consider a subgroup of size 3. We observe that the aggregate trip distance significantly improves if the group does not take (s_4, d_4) into account as (s_4, d_4) is located far away from other source-destination pairs. To address the above mentioned scenario, in this paper, we propose a *subgroup trip planning (SGTP) query* that returns for every subgroup size, the subgroup and the POIs from each required type that have the minimum aggregate trip distance.

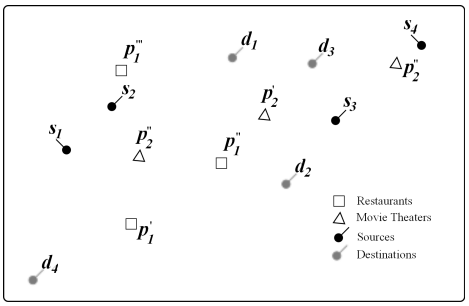


Fig. 1. An example SGTP query, where s_i and d_i represent the source and destination locations of a user u_i, and p_1 and p_2 represent POI types restaurant and movie theater, respectively.

Evaluating an SGTP query requires identifying the subgroup and the POIs from each required type that have the minimum aggregate trip distance for every subgroup size. A straightforward approach to evaluate an SGTP query is to apply our GTP query algorithms [6, 7, 15] for all possible subgroups independently, and select the best subgroup that results in a minimum aggregate distance. This approach incurs high processing overhead and is not scalable as it requires to combinatorially enumerate all possible subgroups and apply our GTP query algorithm for each of these subgroups. We propose an efficient solution to process an SGTP query that can identify the query answer in a single search on the database. Our approach gradually refines the POI search space based on the smallest aggregate trip distance computed using already retrieved POIs, and does not retrieve those POIs from the database that cannot be part of the answer for any subgroup. Furthermore, we develop an efficient technique to identify subgroups of different sizes with the minimum aggregate trip distances for a set of POIs, without enumerating all subgroups.

In summary, the contributions of this paper are summarized as follows:

- We introduce subgroup trip planning (SGTP) queries in spatial databases.
- We propose a hierarchical algorithm to evaluate SGTP queries. We consider minimizing both the total trip distance and the maximum trip distance of the members in the subgroup.
- We develop a technique to compute aggregate trip distances for different subgroup sizes with respect to a set of POIs with reduced computational overhead.

– We evaluate the performance of our approach in experiments using both synthetic and real datasets.

2 Problem Overview

A subgroup trip planning (SGTP) query is formally defined as follows:

Definition 1. ***(SGTP Queries).*** *Given a group G of n users $\{u_1, u_2, \ldots, u_n\}$, the minimum subgroup size n', a set of source locations S, a set of destination locations D, sets of m types of data points $\{D_1, D_2, \ldots, D_m\}$, and an aggregate function f the SGTP query returns for every subgroup size $n'' \in [n', n]$, a subgroup $G' \in G$ of n'' users and a set of data points $P = \{p_1, p_2, \ldots, p_m\}$, $p_t \in D_t$, that minimizes f.*

In this paper, an aggregate function f can be either SUM or MAX, where SUM and MAX return the total or maximum trip distance of group members, respectively. The total trip distance for a subgroup is the summation of each user's trip distance in the subgroup, where a user's trip distance starts from a user u_i's source location s_i and ends at u_i's destination location d_i via the returned data points. On the other hand, the maximum trip distance for a subgroup is computed as the maximum of the trip distances of users in the subgroup.

For an ordered SGTP query, the group fixes the order of the types of data points in which a subgroup visits the data points (e.g., the group may want to visit a shopping center before a restaurant) and the aggregate trip distance is computed accordingly. For a flexible SGTP query, the order of visiting data point types is not fixed, i.e., the group is happy to visit the data points in any order that minimizes the aggregate trip distance.

An SGTP query can be extended to a *k subgroup trip planning (kSGTP) query* that returns for every subgroup size, k subgroups and sets of data points that have k smallest aggregate distances for the group trip. A kSGTP query enables a group to select data points based on other parameters like cost or preference for restaurants. In this paper, we develop solution for processing kSGTP queries.

3 Related Work

GTP queries have been first introduced in [7], where the authors developed algorithms to determine group trips with the minimum total trip distance. In [1, 15], algorithms have been proposed to process ordered GTP queries. Recently, in [6], the authors have proposed an algorithm that can minimize both total and maximum distances of the group members for both ordered and flexible GTP queries. In this paper, we focus on an SGTP query, a new variant of a GTP query. A GTP query identifies only a set of data points whereas an SGTP query returns the subgroup and the set of data points that together minimize the aggregate group trip distance.

Group nearest neighbor (GNN) queries [10, 12–14] and variants [2, 4, 9, 11] have been extensively studied in the literature. A GNN query returns a single type of data point that minimize the aggregate travel distance with respect to the current locations of group members. On the other hand, a GTP query returns a set of data points of

different types that minimize the aggregate travel distance with respect to the source and destination locations of group members. It has been already shown in the literature [7] that extending GNN algorithms for processing SGTP queries would be an exhaustive search and prohibitively computationally expensive.

In [11], Li et al. have proposed a flexible aggregate similarity search that finds the nearest data point and the corresponding subgroup for a fixed subgroup size; for example, a group may query for the nearest data point to 50% of group members. In [2], the authors have developed an approach for spatial consensus queries that find subgroups of different size and corresponding data points that together minimize the aggregate travel distance. In this paper, we introduce a subgroup trip planning query, which is different from the above mentioned variants of GNN queries.

4 Our Approach

In this section, we present our solution to process kSGTP queries. Though it is possible to evaluate an SGTP query by applying the GTP query algorithm [6, 7, 15] independently for all subgroups, this straightforward solution would be an exhaustive search and require to access same data multiple times. More specifically, for every subgroup of size j from a group of n users, this approach requires to apply kGTP query algorithm $\binom{n}{j}$ times to evaluate the answers for $\binom{n}{j}$ groups independently and then select k sets of data points from the evaluated answers and corresponding k sets of j users that together provide k smallest aggregate trip distances. The limitation of this straightforward approach is that the combined query processing overhead is very high and does not scale. We assume that data points of the same type (e.g., restaurants) are indexed using a single R^*-tree [3], and develop an approach to evaluate the query answers for different subgroup sizes *concurrently* with a single traversal on the R^*-trees $R_1, R_2, \ldots, R_m$. Our approach prunes those POIs that cannot be part of the SGTP query answer using the smallest aggregate trip distance computed based on the already retrieved POIs from the database. We develop a technique to identify the subgroups of different sizes with minimum aggregate trip distances without computing aggregate trip distances for all possible subgroups, which further reduces the query processing overhead.

Algorithm 1 shows the steps to evaluate an kSGTP query. The input to the algorithm are a group of n users $G = \{u_1, u_2, \ldots, u_n\}$, source locations $S = \{s_1, s_2, \ldots, s_n\}$, destination locations $D = \{d_1, d_2, \ldots, d_n\}$, m types of data points for $m > 0$, the number k of required sets of data points, the minimum subgroup size n', and an aggregate function f. The output of the algorithm is the answer set $A = \{A_j\}$, where $n' \leq j \leq n$, represents the query answer for subgroup size j. A_j contains k subgroups consisting of j members from G and corresponding data points, $\{G^1, \{p_1^1, p_2^1, \ldots, p_m^1\}\}, \{G^2, \{p_1^2, p_2^2, \ldots, p_m^2\}\}, \ldots, \{G^k, \{p_1^k, p_2^k, \ldots, p_m^k\}\}$ having the k smallest aggregate distances for the group trip.

We use the following notations in our algorithm:

- r_i: a data point or a minimum bounding rectangle (MBR) of a node of R_i.
- $Dist_{min}(.,.)(Dist_{max}(.,.))$: the minimum (maximum) distance between two parameters.

- $d^j_{min}(w_1, w_2, \ldots, w_m)$: the distance computed as the smallest minimum aggregate trip distances computed for all possible subgroups of size j, where the minimum aggregate distance for a subgroup G' of size j is determined as $f_{u_i \in G'}\big(Dist_{min}(s_i, w_1) + \sum_{t=2}^{m-1} Dist_{min}(w_t, w_{t+1}) + Dist_{min}(d_i, w_m)\big)$, where $w_1, w_2, \ldots, w_m$ are data points or MBRs of R^*-tree nodes.
- $d^j_{max}(w_1, w_2, \ldots, w_m)$: the distance computed as the smallest maximum aggregate trip distances computed for all possible subgroups of size j, where the maximum aggregate distance for a subgroup G' of size j is determined as $f_{u_i \in G'}\big(Dist_{max}(s_i, w_1) + sum_{t=2}^{m-1} Dist_{max}(w_t, w_{t+1}) + Dist_{max}(d_i, w_m)\big)$, where $w_1, w_2, \ldots, w_m$ are data points or MBRs of R^*-tree nodes.
- $MinDist[j][k]$: The k^{th} smallest distance for subgroup size j from already computed $d^j_{max}(w_1, w_2, \ldots, w_m)$s.

The algorithm works in a hierarchical manner. The algorithm maintains a priority queue Q_p, which is initially ordered based on the minimum $d^{n'}_{min}$ and reordered based on the minimum $d^{n'+1}_{min}$ when the query answer for subgroup size $n'+1$ has been identified. The reordering continues through $d^{n'+2}_{min}, d^{n'+3}_{min}, \ldots, d^{n}_{min}$ until the query answer for subgroup size n, i.e., the entire group has been identified.

The algorithm first initializes all entries of A with $\emptyset$, $Mindist$ with ∞, cur with n', and end with 0 (Line 1.1). The search for the query answer starts from the root nodes of R^*-trees, $R_1, R_2, \ldots, R_m$; the algorithm inserts the root nodes of $R_1, R_2, \ldots, R_m$ together with $\bigcup_{j=n'}^{n} d^j_{min}$ into a priority queue Q_p. The elements of Q_p are stored in order of $d^{n'}_{min}$. In each iteration of the search, the algorithm dequeues $r_1, r_2, \ldots, r_m$ from Q_p. If all $r_1, r_2, \ldots, r_m$ are data points then the algorithm updates the answers for a subgroup size j, $cur \leq j \leq n$, if the condition $d^j_{min}(r_1, r_2, \ldots, r_m) < MinDist[j][k]$ is satisfied (Lines 1.8–1.9). The algorithm checks this condition because at any point in time Q_p is ordered based on a d^j_{min} and it may happen that $d^{j'}_{min}(r_1, r_2, \ldots, r_m)$ for $j' > j$ is greater than current $MinDist[j'][k]$ and cannot be part of the answer for subgroup size j'.

After updating the answer set, the algorithm checks whether kSGTP answer for subgroup size cur have been already found, i.e., $d^{cur}_{min}(r_1, r_2, \ldots, r_m) > MinDist[cur][k]$. If the next subgroup size, $cur+1$ is greater than n then the algorithm terminates by assigning 1 to the variable end (Line 1.16). Otherwise, the remaining elements of priority queue is reordered based on $d^{cur+1}_{min}(r_1, r_2, \ldots, r_m)$.

On the other hand, if $r_1, r_2, \ldots, r_m$ are not data points, the algorithm computes W using the function $FindSets$. $FindSets$ determines all possible sets $\{w_1, w_2, \ldots, w_m\}$s, where w_j represents either one of the child node of r_j or the data point r_j. In case of an ordered SGTP query, $FindSets$ only computes ordered set of data points/R^*-tree nodes, whereas in case of a flexible SGTP query, $FindSets$ considers all combination of data points/R^*-tree nodes.

For each set $\{w_1, w_2, \ldots, w_m\}$ in W, the algorithm computes $\bigcup_{j=n'}^{n} \{d^j_{min}(w_1, w_2, \ldots, w_m), d^j_{max}(w_1, w_2, \ldots, w_m)\}$ using function $CompTripDist$ (please see Section 4.1). Before inserting $\{w_1, w_2, \ldots, w_m\}$ into Q_p, the algorithm checks whether it is possible to prune $\{w_1, w_2, \ldots, w_m\}$. The algorithm prunes $\{w_1, w_2, \ldots, w_m\}$ if for every $n' \leq j \leq n$, $d^j_{min}(w_1, w_2, \ldots, w_m) > MinDist[j][k]$.

Algorithm 1: kSGTP(G,S,D,n',k,f)

Input : $G=\{u_1,u_2,\ldots,u_n\}$, $S=\{s_1,s_2,\ldots,s_n\}$, $D=\{d_1,d_2,\ldots,d_n\}$, n', k, and f
Output: $A=\{A_{n'},A_{n'+1},\ldots,A_n\}$

1.1 $Initialize(A,MinDist,cur,end)$;
1.2 $Enqueue(Q_p,root_1,root_2,\ldots,root_m,\bigcup_{j=n'}^{n} d_{min}^{j}(root_1,root_2,\ldots,root_m))$;
1.3 **while** Q_p *is not empty and* $end=0$ **do**
1.4 $\quad\{r_1,r_2,\ldots,r_m,\bigcup_{i=n'}^{n} d_{min}^{i}(r_1,r_2,\ldots,r_m)\} \leftarrow Dequeue(Q_p)$;
1.5 $\quad$**if** $r_1, r_2, \ldots, r_m$ *are data points* **then**
1.6 $\quad\quad j \leftarrow cur$;
1.7 $\quad\quad$**while** $j \le n$ **do**
1.8 $\quad\quad\quad$**if** $d_{min}^{j}(r_1,r_2,\ldots,r_m) < MinDist[j][k]$ **then**
1.9 $\quad\quad\quad\quad Update(G,S,D,k,f,A_j,d_{min}^{j},j)$;
1.10 $\quad\quad\quad j \leftarrow j+1$;
1.11 $\quad\quad$**if** $d_{min}^{cur}(r_1,r_2,\ldots,r_m) > MinDist[cur][k]$ **then**
1.12 $\quad\quad\quad cur \leftarrow cur+1$;
1.13 $\quad\quad\quad$**if** $cur \le n$ **then**
1.14 $\quad\quad\quad\quad Reorder(Q_p,cur)$;
1.15 $\quad\quad\quad$**else**
1.16 $\quad\quad\quad\quad end \leftarrow 1$;
1.17 $\quad$**else**
1.18 $\quad\quad W \leftarrow FindSets(r_1,r_2,\ldots,r_m)$;
1.19 $\quad\quad$**for** *each* $(w_1,w_2,\ldots,w_m) \in W$ **do**
1.20 $\quad\quad\quad\bigcup_{j=n'}^{n}\{d_{min}^{j}(w_1,w_2,\ldots,w_m),d_{max}^{j}(w_1,w_2,\ldots,w_m)\} \leftarrow$ $CompTripDist(S,D,n',f,w_1,w_2,\ldots,w_m)$;
1.21 $\quad\quad\quad j \leftarrow n'$;
1.22 $\quad\quad\quad entry \leftarrow 0$;
1.23 $\quad\quad\quad$**while** $j \le n$ **do**
1.24 $\quad\quad\quad\quad$**if** $d_{min}^{j}(w_1,w_2,\ldots,w_m) \le MinDist[j][k]$ **then**
1.25 $\quad\quad\quad\quad\quad$**if** $entry=0$ **then**
1.26 $\quad\quad\quad\quad\quad\quad Enqueue(Q_p,w_1,w_2,\ldots,w_m,\bigcup_{i=n'}^{n} d_{min}^{i}(w_1,w_2,\ldots,w_m))$;
1.27 $\quad\quad\quad\quad\quad\quad entry \leftarrow 1$;
1.28 $\quad\quad\quad\quad\quad$**if** $d_{max}^{j}(w_1,w_2,\ldots,w_m) \le MinDist[j][k])$ **then**
1.29 $\quad\quad\quad\quad\quad\quad Update(MinDist[j],d_{max}^{j}(w_1,w_2,\ldots,w_m))$;
1.30 $\quad\quad\quad\quad j \leftarrow j+1$;
1.31 **return** A;

4.1 Function *CompTripDist*

The purpose of the function *CompTripDist* is to compute d_{min}^{j}s and d_{max}^{j}s for a set of R^*-tree nodes or data points $\{w_1,w_2,\ldots,w_m\}$ with respect to a set of source and destination locations for subgroup size j varying from n' to n. In a straightforward approach to determine these distances, we can first compute each user's individual minimum and

maximum trip distances and then for every subgroup size j, we have to determine the minimum and maximum aggregate trip distances for all possible subgroups of size j and select the minimum and maximum from the computed aggregate trip distances as d^j_{min} and d^j_{max}, respectively. We develop an algorithm that avoids computing aggregate distances for every subgroup and thus, reduces computational overhead.

Algorithm 2: ComputeTripDist($S, D, n', f, w_1, w_2, \ldots, w_m$)

Input : $S = \{s_1, s_2, \ldots, s_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, n', f, and $w_1, w_2, \ldots, w_m$
Output: $\bigcup_{j=n'}^{n} \{d^j_{min}(w_1, w_2, \ldots, w_m), d^j_{max}(w_1, w_2, \ldots, w_m)\}$

2.1 $InterMinDist \leftarrow Dist_{min}(w_1, w_2) + Dist_{min}(w_2, w_3) + \cdots + Dist_{min}(w_{m-1}, w_m)$;
2.2 $InterMaxDist \leftarrow Dist_{max}(w_1, w_2) + Dist_{max}(w_2, w_3) + \cdots + Dist_{max}(w_{m-1}, w_m)$;
2.3 $i \leftarrow 1$;
2.4 **while** $i \leq n$ **do**
2.5 $\quad Enqueue(DMinQ_p, Dist_{min}(s_i, w_1) + Dist_{min}(d_i, w_m))$;
2.6 $\quad Enqueue(DMaxQ_p, Dist_{max}(s_i, w_1) + Dist_{max}(d_i, w_m))$;
2.7 $\quad i \leftarrow i + 1$;
2.8 $j \leftarrow 1$;
2.9 $EndMinDist, EndMaxDist \leftarrow 0$;
2.10 **while** $j \leq n$ **do**
2.11 $\quad EndMinDist \leftarrow Compf(EndMinDist, Dequeue(DMinQ_p), f)$;
2.12 $\quad EndMaxDist \leftarrow Compf(EndMaxDist, Dequeue(DMaxQ_p), f)$;
2.13 $\quad$ **if** $j \geq n'$ **then**
2.14 $\quad\quad$ **if** $f =$ SUM **then**
2.15 $\quad\quad\quad d^j_{min}(w_1, w_2, \ldots, w_m) \leftarrow EndMinDist + j \times InterMinDist$;
2.16 $\quad\quad\quad d^j_{max}(w_1, w_2, \ldots, w_m) \leftarrow EndMaxDist + j \times InterMaxDist$;
2.17 $\quad\quad$ **else**
2.18 $\quad\quad\quad d^j_{min}(w_1, w_2, \ldots, w_m) \leftarrow EndMinDist + InterMinDist$;
2.19 $\quad\quad\quad d^j_{max}(w_1, w_2, \ldots, w_m) \leftarrow EndMaxDist + InterMaxDist$;
2.20 $\quad j \leftarrow j + 1$;
2.21 **return** $\bigcup_{j=n'}^{n} \{d^j_{min}(w_1, w_2, \ldots, w_m), d^j_{max}(w_1, w_2, \ldots, w_m)\}$;

Algorithm 2 shows pseudocode for the function *CompTripDist*. The input to the algorithm are source locations $S = \{s_1, s_2, \ldots, s_n\}$, destination locations $D = \{d_1, d_2, \ldots, d_n\}$, the minimum subgroup size n', an aggregate function f, and a set of R^*-tree nodes or data points $\{w_1, w_2, \ldots, w_m\}$. The output of the algorithm is the set of distances $\bigcup_{j=n'}^{n} \{d^j_{min}(w_1, w_2, \ldots, w_m), d^j_{max}(w_1, w_2, \ldots, w_m)\}$. For $d^j_{min}(w_1, w_2, \ldots, w_m)$, we need to consider j users who have j smallest minimum (maximum) trip distances via $(w_1, w_2, \ldots, w_m)$. The minimum (maximum) distances to travel from w_1 to w_m via $w_2, w_3, \ldots, w_{m-1}$ remain constant for all users' minimum (maximum) trip distances. The trip distances of individual users differ due to their different source and destination locations, i.e., the summation of the minimum (maximum) distance to travel from a user's source location to w_1 and the minimum (maximum) distance to travel from w_m

to the user's destination. Based on this observation, the steps of *CompTripDist* are as follows.

To compute $\bigcup_{j=n'}^{n}\{d_{min}^{j}(w_1, w_2, \ldots, w_m)\}$, the algorithm uses two variables, *InterMinDist* and *EndMinDist*, where *InterMinDist* stores the minimum distance from w_1 to w_m via $w_2, w_3, \ldots, w_{m-1}$ and *EndMinDist* stores the summation of the users' minimum aggregate distance from their sources to w_1 and the users' minimum aggregate distance from w_m to their destinations. The algorithm computes *InterMinDist* in Line 2.1 and *InterMinDist* remains constant for all subgroups of size j.

On the other hand, *EndMinDist* changes based on the source and destination locations of the users in a subgroup. To compute *EndMinDist* for a subgroup size j, we need to determine j smallest values from n distances, where n is the number of users in the group and each distance represents the summation of the minimum distance from a user u_i's source to w_1 and the minimum distance from w_m to u_i's destination, i.e., $Dist_{min}(s_i, w_1) + Dist_{min}(d_i, w_m)$. The algorithm uses a priority queue $DMinQ_p$ to store these n distances in a sorted manner; $DMinQ_p$ is ordered based on $Dist_{min}(s_i, w_1) + Dist_{min}(d_i, w_m)$. We can have j smallest distances for computing *EndMinDist* by dequeuing first j elements from $DMinQ_p$.

In every iteration, the algorithm starts to dequeue a distance from $DMinQ_p$ and updates a variable *EndMinDist* using function $Compf$ based on the aggregate function f. If f is SUM, $Compf$ adds the dequeued distance to *EndMinDist*, and if it is MAX, $Compf$ assigns the dequeued distance to *EndMinDist*, since the current dequeued distance is always greater than previous one dequeued from $DMinQ_p$ (Line 2.11).

In addition, if n' or more distances have been dequeued from $DMinQ_p$, i.e., $j \geq n'$, the algorithm computes corresponding $d_{min}^{j}(w_1, w_2, \ldots, w_m)$ as $EndMinDist + i \times InterMinDist$, if f is SUM. Otherwise, for $f =$ MAX, $d_{min}^{j}(w_1, w_2, \ldots, w_m)$ is computed as $EndMinDist + InterMinDist$.

Similarly, the algorithm computes $\bigcup_{j=n'}^{n}\{d_{max}^{j}(w_1, w_2, \ldots, w_m)$. In summary, our algorithm avoids redundant computations, because we do not need to consider all subgroups while finding the subgroups that provide smallest minimum aggregate trip distances for different subgroup sizes.

5 Experiments

In this section, we present experiments to show the performance of our proposed approach for SGTP queries using both real and synthetic datasets. The real dataset C consists of 62,556 points of interests (i.e., data points) of California. The synthetic datasets U and Z are generated using uniform and a Zipfian distribution, respectively. The total space is normalized into a span of $10,000 \times 10,000$ square units. We use a desktop with a Intel Core 2 Duo 2.40 GHz CPU and 4 GBytes RAM to run our experiments.

We measure the performance of our algorithm in terms of IO costs and the query processing time. We run the experiments for 10 sample SGTP queries and show the average performance. For every SGTP query sample, we randomly select an area in the total space as a query area, and then generate the source and destination locations of group members in the selected query area using a uniform random distribution. We vary the group size, the subgroup size, the number of required sets of data points (k), the

query area, i.e., the minimum bounding rectangle covering the source and destination locations (M), and the dataset size in different sets of experiments. Table 1 shows the range and default value of each parameter. The subgroup size is measured in terms of percentage of the group size; e.g., 60% subgroup size means 38 group members with respect to a default group size 64. We fix the number of required POI types to 2 as a group normally plans a trip for a limited number of POI types. We run the experiments for ordered kSGTP queries as it is a common scenario that the group predefines the order of visiting POI types. Note that our approach is applicable for both flexible and ordered kSGTP queries for any number of POI types.

Parameter	Range	Default
Group size	4, 16, 64, 256	64
Subgroup size	60%, 70%, 80%, 90%	70%
Query area M	2%, 4%, 8%, 16%	4%
k	2, 4, 8, 16	4
Data set size (Synthetic)	5K, 10K, 15K, 20K	-

Table 1. Experiment Setup

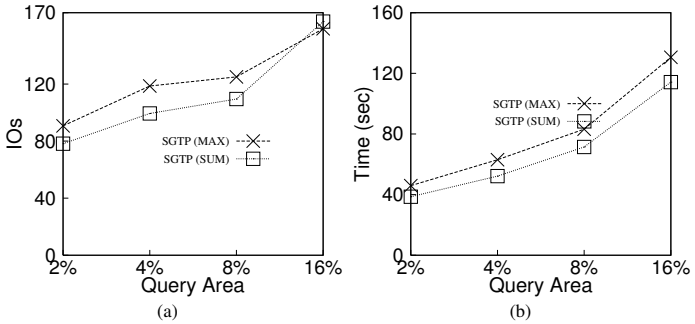
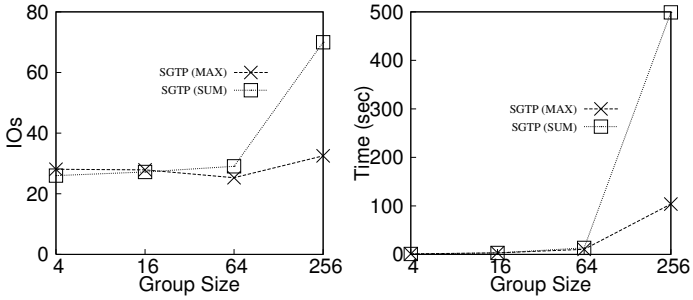


Fig. 2. Effect of query area for kSGTP queries (dataset C)

Effect of Query Area (M): The query area M is varied as 2%, 4%, 8%, and 16% of the data space. The IO cost and processing time for processing SGTP queries are measured for both aggregate SUM and MAX functions (Figures 2(a) and 2(b)). Both the IO cost and the processing time increase with the increase of the area M as for a larger M we need to access more data points from R*-trees than that of a smaller M for the aggregate SUM and MAX functions.

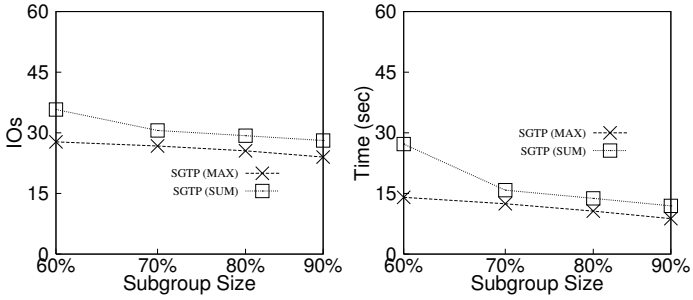
Effect of Group Size: We vary the group size as 4, 16, 64, and 256, and measure IOs and processing time for SGTP queries for aggregate functions SUM and MAX. Figure 3(b) shows that the processing time increases with the increase of the group size for both MAX and SUM functions. The reason behind this is for a larger group size, SGTP requires to process a large number of subgroups. Thus, it computes the aggregate trip distances for these increasing number of subgroups. However, the IOs remain almost constant (MAX) or slightly increase (SUM) with the increase of the group size (see Figure 3(a)).

Effect of Subgroup Size: In this set of experiments, we vary the subgroup size as 60%, 70%, 80%, and 90% of the default group size of 64, and measure IOs and processing time for both aggregate functions SUM and MAX. In Figure 4(a), we can see



(a) (b)

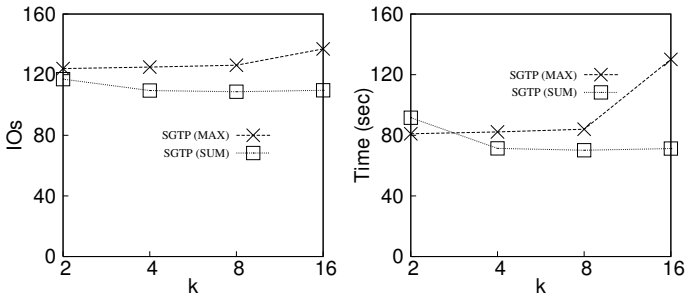
Fig. 3. Effect of group size for *k*SGTP queries (dataset *C*)



(a) (b)

Fig. 4. Effect of subgroup size for *k*SGTP queries (dataset *C*)

that the IOs remain constant with the increase of the subgroup size. However, there is a decrease in processing time with the increase of the subgroup size (see Figure 4 (b)). As the subgroup size increases, the number of subgroups decreases and hence, less computations are required in computing the aggregate trip distances.



(a) (b)

Fig. 5. Effect of *k* for *k*SGTP queries (dataset *C*)

Effect of k: In this set of experiments, *k* is varied as 2, 4, 8, and 16. In Figure 5(a), we observe that the IO cost slightly increases or remains constant with the increase of *k* for both SUM and MAX aggregate functions. Figure 5(b) shows that the processing

time remains constant (with a slight decrease at initial phase) for SUM and increases for MAX with the increase of k.

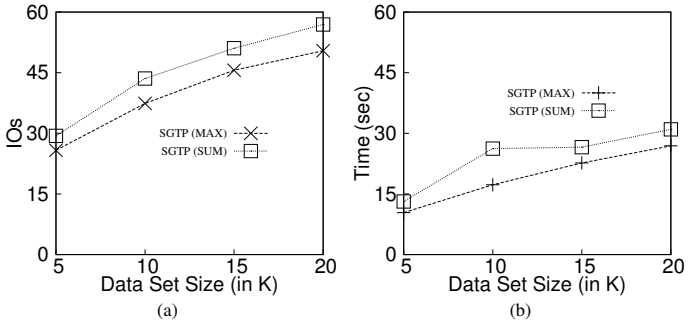


Fig. 6. Effect of dataset size for kSGTP queries (dataset U)

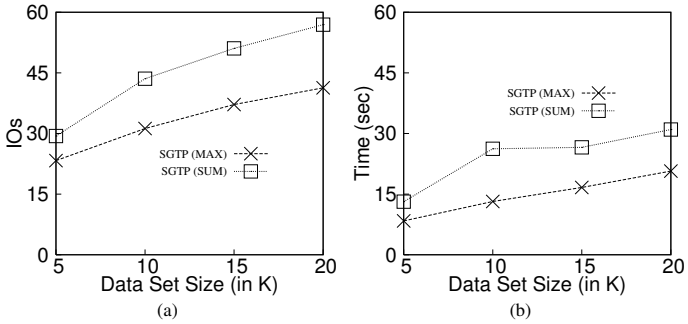


Fig. 7. Effect of dataset size for kSGTP queries (dataset Z)

Effect of Data Set Size: In this set of experiments, we vary the dataset size as 5K, 10K, 15K, and 20K for both uniform (U) and Zipfian (Z) distributions. Figures 6(a) and 6(b) show the IO cost and processing time, respectively, for different data set sizes with U distribution for aggregate functions SUM and MAX. The experimental results show that SGTP requires more IOs and processing time for SUM aggregate function than MAX aggregate function. Figures 7(a) and 7(b) show the IO cost and processing time required by SGTP, respectively, for different data set sizes with Z distribution. The results show similar characteristics of U distribution.

6 Conclusion

In this paper, we introduced subgroup trip planning (SGTP) queries that enable a group to identify the subgroups and POI sets that together minimize the total or maximum trip distance. We have developed a hierarchical approach to process SGTP queries. Our approach avoids the computation of aggregate trip distances independently for different subgroups and thus, reduces computational overhead. Our experiments also show that our algorithm can evaluate a SGTP query with reduced processing time. In the future,

we plan to develop algorithms for processing SGTP queries for road networks and the obstructed space. We also aim to protect location privacy [5, 8] of group members for SGTP queries.

Acknowledgments

This research is partially supported by the ICT Division - Government of the People's Republic of Bangladesh.

References

1. Elham Ahmadi and Mario A. Nascimento. A mixed breadth-depth first search strategy for sequenced group trip planning queries. In *MDM*, pages 24–33, 2015.
2. Mohammed Eunus Ali, Egemen Tanin, Peter Scheuermann, Sarana Nutanong, and Lars Kulik. Spatial consensus queries in a collaborative environment. *ACM Trans. Spatial Algorithms and Systems*, 2(1):3, 2016.
3. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.
4. Ke Deng, Shazia Wasim Sadiq, Xiaofang Zhou, Hu Xu, Gabriel Pui Cheong Fung, and Yansheng Lu. On group nearest group query processing. *IEEE TKDE*, 24(2):295–308, 2012.
5. Tanzima Hashem, Mohammed Eunus Ali, Lars Kulik, Egemen Tanin, and Anthony Quattrone. Protecting privacy for group nearest neighbor queries with crowdsourced data and computing. In *UbiComp*, pages 559–562, 2013.
6. Tanzima Hashem, Sukarna Barua, Mohammed Eunus Ali, Lars Kulik, and Egemen Tanin. Efficient computation of trips with friends and families. In *CIKM*, pages 931–940, 2015.
7. Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, and Lars Kulik. Group trip planning queries in spatial databases. In *SSTD*, pages 259–276, 2013.
8. Tanzima Hashem and Lars Kulik. Safeguarding location privacy in wireless ad-hoc networks. In *Ubicomp*, pages 372–390, 2007.
9. Tanzima Hashem, Lars Kulik, and Rui Zhang. Privacy preserving group nearest neighbor queries. In *EDBT*, pages 489–500, 2010.
10. Feifei Li, Bin Yao, and Piyush Kumar. Group enclosing queries. *IEEE TKDE*, 23(10):1526–1540, 2011.
11. Yang Li, Feifei Li, Ke Yi, Bin Yao, and Min Wang. Flexible aggregate similarity search. In *SIGMOD*, pages 1009–1020, 2011.
12. Sansarkhuu Namnandorj, Hanxiong Chen, Kazutaka Furuse, and Nobuo Ohbo. Efficient bounds in finding aggregate nearest neighbors. In *DEXA*, pages 693–700, 2008.
13. Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. Group nearest neighbor queries. In *ICDE*, page 301, 2004.
14. Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun K. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2):529–576, 2005.
15. Samiha Samrose, Tanzima Hashem, Sukarna Barua, Mohammed Eunus Ali, Mohammad Hafiz Uddin, and Md. Iftekhar Mahmud. Efficient computation of group optimal sequenced routes in road networks. In *MDM*, pages 122–127, 2015.
16. Da Yan, Zhou Zhao, and Wilfred Ng. Efficient processing of optimal meeting point queries in euclidean space and road networks. *Knowl. Inf. Syst.*, 42(2):319–351, 2015.