Group Trip Planning Queries in Spatial Databases

CONFERENCE PAPER · JANUARY 2013

DOI: 10.1007/978-3-642-40235-7_15

Tanzima Hashem
Bangladesh University of Engineering and …
12 PUBLICATIONS **84** CITATIONS
SEE PROFILE

Tahrima Hashem
University of Dhaka
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Lars Kulik
University of Melbourne
100 PUBLICATIONS **1,301** CITATIONS
SEE PROFILE

Group Trip Planning Queries in Spatial Databases

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

[1] Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh,
{tanzimahashem, eunus}@cse.buet.ac.bd
[2] Department of Computer Science and Engineering
Dhaka University, Dhaka, Bangladesh,
tahrimacsedu14@gmail.com
[3] Department of Computing and Information System
University of Melbourne, VIC 3010, Australia,
lkulik@unimelb.edu.au

Abstract. Location-based social networks grow at a remarkable pace. Current location-aware mobile devices enable us to access these networks from anywhere and to connect to friends via social networks in a seamless manner. These networks allow people to interact with friends and colleagues in a novel way, for example, they may want to spontaneously meet in the next hour for dinner at a restaurant nearby followed by a joint visit to a movie theater. This motivates a new query type, which we call a group trip planning (GTP) query: the group has an interest to minimize the total travel distance for all members, and this distance is the sum of each user's travel distance from each user's start location to destination via the restaurant and the movie theater. Formally, for a set of user source-destination pairs in a group and different types of data points (e.g., a movie theater versus a restaurant), a GTP query returns for each type of data points those locations that minimize the total travel distance for the entire group. We develop efficient algorithms to answer GTP queries, which we show in extensive experiments.

Keywords: Group nearest neighbor queries, Group trip planning queries, Location-based services, Location-based social networks, Spatial Databases

1 Introduction

Location-based social networks such as Facebook [1], Google+ [2], and Loopt [3] enable a group of friends to remain connected from virtually anywhere at any time via location aware mobile devices. In these networks users can share their locations with others and interact with other users in a novel way. For example, a group of friends may want to spontaneously meet in the next hour for dinner at a restaurant nearby followed by a joint visit to a movie theater. The users are typically at different places, e.g., on a late afternoon all of them may still be at their offices and workplaces. Before going home in the evening, they would still like to organise a group dinner and movie event. Usually, all users would like to meet at a restaurant and movie theater that is nearby. To enable users to arrange a trip with a minimum total travel distance, we introduce a new type of query called a *group trip planning (GTP) query*.

Specifically, for a set of user source-destination pairs in a group and different types of data points (e.g., a movie theater versus a restaurant), a GTP query returns for each type of data points those locations that minimize the total travel distance for the entire group. The total travel distance of a group is the sum of each user's travel distance from source to destination via the data points. Figure 1 shows an example, where the pair of data points (p_1'', p_2') minimizes the total travel distance for the group trip. The group may fix the order of the planned locations (e.g., first the movie theater, then the restaurant) or keep the order flexible (e.g., the group is happy to visit the movie theater and the restaurant in any order). We call the former type *ordered GTP queries* and the latter *flexible GTP queries*. In this paper, we develop algorithms to evaluate both type of GTP queries.

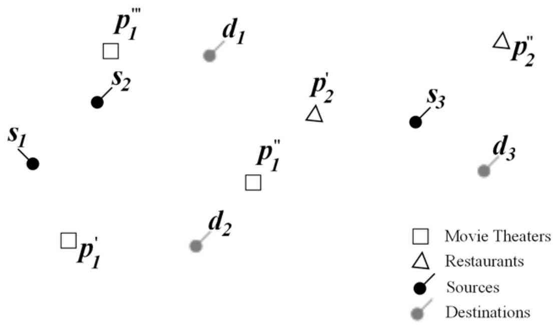


Fig. 1. An example scenario.

Query processing for a group of users instead of a single user involves high computational overhead because this query has to be evaluated with respect to a set of locations instead of a single location. Existing studies [4, 5] show that iteratively applying algorithms that are designed for a single user incur large query processing overheads if they are used to evaluate group queries. Thus, the development of efficient algorithms for processing group nearest neighbor (GNN) queries [4, 5] and its variants [6, 7] has recently evolved into an important research area. A GNN query finds the data point that minimizes the aggregate distance with respect to the locations of the users in the group. In this paper, we address group trip planning queries. Though trip planning queries for a single user have been addressed [8, 9] in the literature, GTP queries have not yet been explored.

A GTP query can be mapped to a GNN query, if a group meets at a single location during their trip; existing GNN algorithms can be applied to the GTP query to find the data point that minimizes the total travel distance for a set of source and destination locations of the group members. However, if the group stops at least at two different types of locations, we show that applying GNN algorithms [4, 5] to evaluate GTP queries increases the query processing cost significantly. Such a technique evaluates GNN queries multiple times and requires multiple independent searches on the database. We develop an efficient algorithm for processing GTP queries that finds the data points with a single traversal on the database. The key idea of our algorithm is to reduce the search space, i.e., to avoid the computation of total travel distances for data points that cannot be part of the query answer. We develop a set of pruning techniques to eliminate the data points from the search while evaluating GTP queries.

In summary, we make the following contributions:

– We propose a new type of query, the group trip planning (GTP) query and propose the first solution to process the query.
– We develop an efficient algorithm to evaluate GTP queries. Our algorithm can evaluate both ordered and flexible GTP queries.
– We perform an extensive experimental study to show the efficiency of our proposed algorithms, including a detailed comparative analysis.

The rest of the paper is organized as follows. Section 2 presents the problem setup and Section 3 reviews existing works related to this problem. In Section 4, we propose our algorithms to evaluate GTP queries and in Section 5, we present extensive experiments to validate the efficiency of our algorithms. Section 6 concludes the paper with future research directions.

2 Problem Overview

In a group trip planning (GTP) query, a group of users specify their source and destination locations, the types of data points that they want to visit together while traveling from their source to destination locations. For a group of n users let S represent the set of source locations $\{s_1, s_2, \ldots, s_n\}$ and D the set of destination locations $\{d_1, d_2, \ldots, d_n\}$. The source and destination locations of a user u_i are denoted as s_i and d_i, respectively. A set of data points of type t in a 2-dimensional space is denoted by D_t. If a group plans a trip with m different types of data points along their trip, the GTP query returns a set of data points $\{p_1, p_2, \ldots, p_m\}$, $p_t \in D_t$, such that an aggregate function $f(S, D, p_1, p_2, \ldots, p_m)$ is minimal. The function $f(S, D, p_1, p_2, \ldots, p_m)$ represents the total travel distance for the group trip as $\left(\sum_{i=1}^{n} Dist(s_i, p_1) + n \times \sum_{i=1}^{m-1} Dist(p_i, p_{i+1}) + \sum_{i=1}^{n} Dist(d_i, p_m)\right)$, where $Dist(.,.)$ is the Euclidean distance between two locations. In general, a group trip planning (GTP) query is formally defined as follows:

Definition 1. ***(Group Trip Planning (GTP) Queries).*** *Given 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 GTP query returns a set of data points $\{p_1, p_2, \ldots, p_m\}$, $p_t \in D_t$, that minimizes f.*

For a set of data points $\{p_1, p_2, \ldots, p_m\}$, in an ordered GTP query, the group determines the order in which the group plans to visit the m data points. The parameters $p_1, p_2, \ldots, p_m$ in the aggregate function f are passed in the order specified by the group. In a flexible GTP query, however f is evaluated for every possible order of the data points $p_1, p_2, \ldots, p_m$.

A group may be interested in k sets of data points $\{p_1^1, p_2^1, \ldots, p_m^1\}$, $\{p_1^2, p_2^2, \ldots, p_m^2\}$, $\ldots$, $\{p_1^k, p_2^k, \ldots, p_m^k\}$ that have the k smallest total travel distances for the group trip. The group then select one set by considering other factors such as cost and recommendations. If a group queries for k sets of data points for a group trip then the query is called a *k group trip planning (kGTP) query*. In Section 4, we develop algorithms to evaluate kGTP queries.

The symbols used in this paper are summarized in Table 1.

Symbol	Meaning
s_i and d_i	Source and destination locations of a user u_i
$S = \{s_1, s_2, \ldots, s_n\}$	A set of source locations of n users in the group
$D = \{d_1, d_2, \ldots, d_n\}$	A set of destination locations of n users in the group
$\{p_1^k, p_2^k, \ldots, p_m^k\}$	A set of data points that has k^{th} minimum distance for the group trip
$Dist(.,.)$	The Euclidean distance between two locations
$f(S, D, p_1, p_2, \ldots, p_m)$	A function that returns the total travel distance as $\{\sum_{i=1}^{n} Dist(s_i, p_1) + n \times Dist(p_1, p_2) + \cdots + n \times Dist(p_{m-1}, p_m) + \sum_{i=1}^{n} Dist(d_i, p_m)\}$

Table 1. Symbols

3 Related Work

Most of the existing techniques for processing spatial queries assume that data points are indexed, e.g., using an R-tree [10] or its variant R^*-tree [11]. In an R-tree, nearby objects are grouped together with minimum bounding rectangles (MBRs). These MBRs are organized in a hierarchical way such that the root MBR covers the whole data space and the MBR of a parent node covers the MBRs of all of its children. In this paper, we use separate R^*-trees, to index different types of data points.

The most popular query type in spatial database is nearest neighbor queries that return the nearest data point with respect to a given location. A well known approach to evaluate the k NNs is to traverse the R^*-tree in a best-first (BF) [12] manner, which is called the best-first search (BFS). In BFS, the search starts from the root of the tree, and the child nodes are recursively accessed in the increasing order of their distances from the query point. The search process terminates as soon as the k nearest data points are retrieved from the tree.

A group nearest neighbor (GNN) query [4, 5, 13], a variant of the NN query, finds the nearest data point with respect to all user locations of the group. A GNN query minimizes the aggregate distance for the group, where an aggregate distance is measured as the total, minimum or maximum distance of the data point from the group. Papadias et al. [4, 5] have proposed three techniques: multiple query method (MQM), single point method (SPM), and minimum bounding method (MBM), to evaluate GNN queries. All of these three variants use the BFS technique to traverse the R^*-tree. Among these three techniques, MBM performs the best as it traverses the R^*-tree once and takes the area covering the users' location into account. We use MBM for evaluating GNN queries in our approach. In [13], Li et al. have developed exact and approximation algorithms for GNN queries that minimize the maximum distance of the group. In this paper, we require GNN algorithms that minimize the total travel distance.

Recently, some variants of GNN queries have been proposed [6, 7]. In [6], Deng et al. have proposed group nearest group (GNG) query that finds a subset of data points, as opposed to a single data point in GNN, from the dataset such that the aggregate distance from the group to the subset is minimum. The aggregate distance is computed as the summation of all Euclidean distances between a query point and the nearest data point in the subset. In [7], 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 this paper, we introduce a group based trip planning query, which is different from the above mentioned variants.

Trip planning queries [9, 8, 14, 15] have been studied in the literature with respect to a single user. In [9], Li et al. have developed approximation algorithms for finding a set of data points that minimize the total travel distance for the trip. The travel distance starts from the source location, passes through the set of data points, and ends at the destination location of the user, where each data point corresponds to a type (e.g., a restaurant) specified by the user. In [8], Sharifzadeh et al. have developed algorithms to evaluate a optimal sequenced route (OSR) query that returns a route with the minimum length passing through a set of data points in a particular order from the source location of a user, where both order and type of data points are specified by the user. Chen et al. [14] have proposed a generalization of the trip planning query, called multi-rule partial sequenced route (MRPSR) query. A MRPSR query provides a uniform framework to evaluate both of the above mentioned variants [9, 8] of trip planning queries. All these existing techniques assume a single user, and thus are not suitable for a group trip.

A large body of research works focus on developing algorithms for processing a route planing query and its variants [16–19]. A route planning query finds a suitable route that minimizes the desire function such as travel time, shortest path, cost, etc. for *a single source and destination pair*. Route planing queries do not include data points in the route and are applicable for a single user. On the other hand, we propose an efficient solution for a *k*GTP query that finds different types of data points for a group trip that minimize the total travel distance of the group.

4 Algorithms

In this section, we present algorithms to process *k*GTP queries. A straightforward way to evaluate *k*GTP queries would be applying a trip planning algorithm [14] for every user in the group independently and determining the group travel distance for every computed set of data points. The process continues to incrementally determine the sets of data points that minimize the total travel distance of every user in the group until the set of data points with the minimum group travel distance have been identified. However, this straightforward solution is not scalable and incurs very high query processing overhead as same data points are accessed by multiple users separately. Instead, we first propose an iterative algorithm as baseline method that does not evaluate the trip planing query independently for every user but still requires to access same data multiple times.

To avoid the limitation of the baseline method, we develop an efficient hierarchical algorithm that evaluates kGTP queries with a single traversal of the database and incurs less processing overhead. The input to the algorithm for a group of n users are source locations $S = \{s_1, s_2, \dots, s_n\}$, destination locations $D = \{d_1, d_2, \dots, d_n\}$, m types of data points for $m > 0$, and the number k of required sets of data points. The output of the algorithm is A that consists of k sets of data points $\{p_1^1, p_2^1, \dots, p_m^1\}, \{p_1^2, p_2^2, \dots, p_m^2\}, \dots, \{p_1^k, p_2^k, \dots, p_m^k\}$ having the k smallest total travel distances for the group trip. We assume that each type of data points are indexed using a separate R^*-tree [11] in the database. Let $R_1, R_2, \dots, R_m$ represent the R^*-trees to index the set of data points in $D_1, D_2, \dots, D_m$, respectively. Note that, our algorithms can also be adopted if a single R^*-tree is used to index all types of data points. We present our two approaches, i.e., iterative and hierarchical, for processing kGTP queries in Section 4.1 and Section 4.2, respectively.

4.1 Iterative Approach

The basic idea of our iterative approach to evaluate a kGTP query is to use group nearest neighbor (GNN) queries. A kGNN query returns the locations of k data points that have the k smallest total travel distances for the group. For ease of understanding, we start the discussion of the iterative algorithm for the number of data type, $m = 1$, $m = 2$, and then generalize the algorithm for any value of m.

For $m = 1$, the iterative approach can evaluate the kGTP query answer with a single iteration using any existing kGNN algorithm [4, 5]. The iterative approach determines the kGTP query answer $p_1^1, p_1^2, \dots, p_1^k$ as k GNNs (group nearest neighbors) from D_1 with respect to $\{S \cup D\}$.

Consider the case $m = 2$, where the group specifies the order of visit as $\{1, 2\}$, i.e., the group visits the data point of type 1 before the data point of type 2. The iterative approach determines the 1^{st} GNN as p_1 from D_1 with respect to S and the 1^{st} GNN as p_2 from D_2 with respect to $\{p_1 \cup D\}$. For the pair of data points p_1 and p_2, the iterative algorithm determines the total travel distance for the group trip. Then the algorithm determines the 2^{nd} GNN as p_2' from D_2 with respect to $\{p_1 \cup D\}$ and computes the total travel distance for the group trip for p_1 and p_2'. The process continues until the k^{th} group nearest data point as p_2 from D_2 with respect to $\{p_1 \cup D\}$ has been determined.

In the next iteration, the algorithm determines the 2^{nd} GNN as p_1 from D_1 with respect to S and repeats the same procedure mentioned above for the 1^{st} GNN from D_1. The iteration continues to incrementally determine the GNNs from D_1 with respect to S until the algorithm finds k pairs of data points that have k smallest total travel distances for the group trip.

Algorithm 1 shows the pseudocode to iteratively evaluate the ordered kGTP query for 2 stops, i.e., $m = 2$. Function $FindGNN(l, S)$ is used in Algorithm 1 to compute the l^{th} GNN that has l^{th} smallest total distance with respect to a set of points S.

In each iteration ($l = 1, 2, \dots$), the algorithm computes k pairs of data points using $FindGNN$ (see Lines 1.5 - 1.12). For each pair p_1 and p_2, p_1 is evaluated as the l^{th} GNN of S and p_2 is evaluated as j^{th} GNN of $\{p_1 \cup D\}$, where j ranges from 1 to k. For each

Algorithm 1: 2S-Ordered-kGTP-IA(S, D, k)

```
     Input  : $S = \{s_1, s_2, \ldots, s_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $k$.
     Output: $A = \{\{p_1^1, p_2^1\}, \{p_1^2, p_2^2\}, \ldots, \{p_1^k, p_2^k\}\}$.
1.1  $A \leftarrow \emptyset$
1.2  $MinDist[1..k] \leftarrow \{\infty\}$
1.3  $l \leftarrow 1$
1.4  repeat
1.5      $p_1 \leftarrow FindGNN(l, S)$
1.6      $j \leftarrow 1$
1.7      while $j \leq k$ do
1.8          $p_2 \leftarrow FindGNN(j, \{p_1 \cup D\})$
1.9          $CurrentDist \leftarrow f(S, D, p_1, p_2)$
1.10         if $CurrentDist \leq MinDist[k]$ then
1.11             $Update(CurrentDist, MinDist, A)$
1.12         $j \leftarrow j + 1$
1.13     $l \leftarrow l + 1$
1.14 until $\sum_{i=1}^{n} Dist(s_i, p_1) \leq MinDist[k]$
1.15 return $A$
```

of these k pairs of data points, the algorithm determines the total travel distance for the group trip as $CurrentDist$. We use an array $MinDist[1..k]$ of k entries to prune the pair of data points that cannot be the part of the answer set A and to check the termination condition of the algorithm. Each entry $MinDist[i]$ represents the i^{th} smallest distance of total travel distances for the group trip computed so far. The entries of $MinDist$ are initialized to ∞ and then checked for update after every computation of $CurrentDist$. If $CurrentDist \leq MinDist[k]$, p_1 and p_2 become one of the k pairs that have k smallest total travel distance among the explored pairs of data points so far and the function $Update$ is called to update $MinDist$ and A (Line 1.11).

The parameter l is initialized to 1 and is incremented by 1 at every iteration until k pairs of data points that minimize the total travel distances for the group trip have been determined. We use the following lemma to check the terminating condition of Algorithm 1.

Lemma 1. *Let p_1 be the l^{th} group nearest neighbor with respect to the set of source locations S and $MinDist[k]$ be the k^{th} smallest total travel distances for the group trip computed so far. The k pairs of data points that minimize the total travel distance for the group trip have been found by the algorithm if $\sum_{i=1}^{n} Dist(s_i, p_1) > MinDist[k]$.*

Proof. For any t^{th} group nearest neighbor p_1' with respect to S, where $t > l$, we have $\sum_{i=1}^{n} Dist(s_i, p_1') > \sum_{i=1}^{n} Dist(s_i, p_1) > MinDist[k]$. Thus, if the condition $\sum_{i=1}^{n} Dist(s_i, p_1) > MinDist[k]$ becomes true then there can be no other unexplored pair of data points that can further minimize the total travel distance for the group trip. □

The proof of Lemma 1 also shows the correctness of our proposed nested algorithm, i.e., the answer set, A includes k pairs of data points that have k smallest total travel distances with respect to S and D of a group.

```
Algorithm 2: 3S-$k$GTP-IA($S,D,k$)
     Input  : $S = \{s_1, s_2, \ldots, s_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $k$.
     Output: $A = \{\{p_1^1, p_2^1, p_3^1\}, \{p_1^2, p_2^2, p_3^2\}, \ldots, \{p_1^k, p_2^k, p_3^k\}\}$.
2.1  $A \leftarrow \emptyset$
2.2  $MinDist[1..k] \leftarrow \{\infty\}$
2.3  $l \leftarrow 1$
2.4  repeat
2.5      $p_1 \leftarrow FindGNN(l, S)$
2.6      $u \leftarrow 1$
2.7      repeat
2.8          $p_2 \leftarrow FindNN(u, p_1)$
2.9          $v \leftarrow 1$
2.10         while $v \leq k$ do
2.11             $p_3 \leftarrow FindGNN(j, \{p_2 \cup D\})$
2.12             $CurrentDist \leftarrow f(S, D, p_1, p_2, p_3)$
2.13             if $CurrentDist \leq MinDist[k]$ then
2.14                 $Update(CurrentDist, MinDist, A)$
2.15             $v \leftarrow j + 1$
2.16         $u \leftarrow v + 1$
2.17     until $\sum_{i=1}^{n} Dist(s_i, p_1) + n \times Dist(p_1, p_2) \leq MinDist[k]$
2.18     $l \leftarrow l + 1$
2.19 until $\sum_{i=1}^{n} Dist(s_i, p_1) \leq MinDist[k]$
2.20 return $A$
```

Algorithm 1 also works for flexible GTP query with slight modifications. For flexible GTP query we have to execute Algorithm 1 twice considering both orders, first p_1 then p_2 and the reverse order, i.e., first p_2 and then p_1. The evaluation of kGTP queries for each order computes k group trips; among these $2k$ group trips we have to select k group trips with k smallest total travel distances.

Consider the case for $m = 3$. Assume that the group specifies the order of visit as $\{1, 2, 3\}$. Algorithm 2 shows the pseudocode for evaluating ordered kGTP queries for $m = 3$. Algorithm 1 works in a similar way to Algorithm 1. The only difference for Algorithm 2 is the addition of an intermediary step using existing nearest neighbor algorithms [20, 12]. We use function *FindNN* to evaluate nearest neighbors (Line 3.8). Similarly, Algorithm 2 can be generalized for ordered kGTP queries with m types of data points by adding $m-2$ intermediary steps using *FindNN*. For flexible kGTP queries with m types of data points, the query evaluation requires to consider all possible combinations of orderings of the data points and adds additional query processing overheads.

For $m \geq 2$, the limitation of the iterative approach is its high computational overhead. To determine the sets of data points that minimize the total travel distance for the group trip, the iterative approach requires to traverse the same data sets multiple times and the overhead increases with the increase of m.

4.2 Hierarchical Approach

In this section, we present an efficient hierarchical approach to evaluate a kGTP query in a single traversal of R^*-trees $R_1, R_2, \ldots, R_m$. The base idea of our hierarchical algorithm, GTP-HA, is to exploit the hierarchical properties of R^*-trees and use a modified best first search (BFS) on $R_1, R_2, \ldots, R_m$ to find k sets of data points that minimize the total travel distances for the group trip.

Algorithm 3: kGTP_HA(S, D, k)

```
     Input  : $S = \{s_1, s_2, \ldots, s_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $k$.
     Output: $A = \{\{p_1^1, p_2^1, \ldots, p_m^1\}, \{p_1^2, p_2^2, \ldots, p_m^2\}, \ldots, \{p_1^k, p_2^k, \ldots, p_2^m\}\}$.
3.1  $end \leftarrow 0$
3.2  $A \leftarrow \emptyset$
3.3  $MinDist[1..k] \leftarrow \{\infty\}$
3.4  $i \leftarrow 1$
3.5  $Enqueue(Q_p, root_1, root_2, \ldots, root_m, 0, d_{max}(root_1, root_2, \ldots, root_m))$
3.6  while $Q_p$ is not empty and $end = 0$ do
3.7      $\{r_1, r_2, \ldots, r_m, d_{min}(r_1, r_2, \ldots, r_m), d_{max}(r_1, r_2, \ldots, r_m)\} \leftarrow Dequeue(Q_p)$
3.8      if $r_1$, $r_2$, ..., $r_m$ are data points then
3.9          $\{p_1^i, p_2^i, \ldots, p_m^i\} \leftarrow \{r_1, r_2, \ldots, r_m\}$
3.10         if $i = k$ then
3.11             $end \leftarrow 1$
3.12         $i \leftarrow i + 1$
3.13     else
3.14         $W \leftarrow FindSets(r_1, r_2, \ldots, r_m)$
3.15         for each $(w_1, w_2, \ldots, w_m) \in W$ do
3.16             Compute $d_{min}(w_1, w_2, \ldots, w_m)$ and $d_{max}(w_1, w_2, \ldots, w_m)$
3.17             if $d_{min}(w_1, w_2, \ldots, w_m) \leq MinDist[k]$ then
3.18                 $Enqueue(Q_p, w_1, w_2, \ldots, w_m, d_{min}(w_1, w_2, \ldots, w_m), d_{max}(w_1, w_2, \ldots, w_m))$
3.19                 if $d_{max}(w_1, w_2, \ldots, w_m) \leq MinDist[k]$ then
3.20                     $Update(MinDist, d_{max}(w_1, w_2, \ldots, w_m))$
3.21 return $A$
```

Algorithm 3 shows the steps for GTP-HA. The notations that we have used for hierarchical approach are summarized below:

- r_j: a data point or a minimum bounding rectangle of a node of R_j.
- $\{w_1, w_2, \ldots, w_m\}$: A set of entities, where each entity w_j represents a data point r_j or a minimum bounding rectangle of a child node of r_j.
- $Dist_{min}(.,.)(Dist_{max}(.,.))$: a function that returns the minimum (maximum) distance between two parameters, where a parameter can be either a point or a minimum bounding rectangle of a R^*-tree node.
- $d_{min}(w_1, w_2, \ldots, w_m)$: the distance computed as $\sum_{i=1}^{n} Dist_{min}(s_i, w_1) + n \times Dist_{min}(w_1, w_2) + \cdots + n \times Dist_{min}(w_{m-1}, w_m) + \sum_{i=1}^{n} Dist_{min}(d_i, w_m)$

- $d_{max}(w_1,w_2,\dots,w_m)$: the distance computed as $\sum_{i=1}^{n} Dist_{max}(s_i,w_1) + n \times Dist_{max}(w_1,w_2) + \cdots + n \times Dist_{max}(w_{m-1},w_m) + \sum_{i=1}^{n} Dist_{max}(d_i,w_m)$
- $MinDist[k]$: The k^{th} smallest distance of already computed $d_{max}(w_1,w_2,\dots,w_m)$s.

The algorithm starts the search from the root nodes of $R_1,R_2,\dots,R_m$. The algorithm inserts the root nodes of $R_1,R_2,\dots,R_m$ together with their d_{min} and d_{max} into a priority queue Q_p. The elements of Q_p are ordered in order of d_{min}. In each iteration of the search, the algorithm dequeues $r_1,r_2,\dots,r_m$ from Q_p. If all $r_1,r_2,\dots,r_m$ are data points then the data points are added to A (Line 3.9). Otherwise, the algorithm computes W using the function $FindSets$. The input parameters of $FindSets$ are $r_1,r_2,\dots,r_m$. $FindSets$ determines all possible sets $\{w_1,w_2,\dots,w_m\}$s, where w_j represents either one of the child node of r_j or the data point r_j. In case of ordered GTP query, $FindSets$ only computes ordered set of data points/R^*-tree nodes, whereas in case of flexible GTP query, $FindSets$ considers all combination of data points/R^*-tree nodes. Consider an example, where r_1,r_2,r_3 represent data points. If a group specifies the order of visiting types of data points as $\{3,1,2\}$ then the computed set is $\{r_3,r_1,r_2\}$, i.e., $w_1 = r_3$, $w_2 = r_1$, and $w_3 = r_2$. If the group is flexible then the computed sets are $\{r_1,r_2,r_3\}$, $\{r_1,r_3,r_2\}$, $\{r_2,r_1,r_3\}$, $\{r_2,r_3,r_1\}$, $\{r_3,r_1,r_2\}$, and $\{r_3,r_2,r_1\}$.

For each set $\{w_1,w_2,\dots,w_m\}$ in W, the algorithm computes $d_{min}(w_1,w_2,\dots,w_m)$ and $d_{max}(w_1,w_2,\dots,w_m)$. Similar to the iterative approach, we use an array $MinDist[1..k]$ to check whether R^*-tree nodes/data points can be pruned using the following lemma:

Lemma 2. *A set of data points or R^*-tree nodes $\{w_1,w_2,\dots,w_m\}$ can be pruned if $d_{min}(w_1,w_2,\dots,w_m) > MinDist[k]$.*

If the condition of Lemma 2 is satisfied for any set $\{w_1,w_2,\dots,w_m\}$, then $\{w_1,w_2,\dots,w_m\}$ or any set computed from the child nodes of $w_1,w_2,\dots,w_m$ can never be part of A. If the condition of Lemma 2 is not satisfied for $\{w_1,w_2,\dots,w_m\}$, i.e., $d_{min}(w_1,w_2,\dots,w_m) \leq MinDist[k]$, then the algorithm inserts $(w_1,w_2,\dots,w_m)$ into Q_p and updates $MinDist$ if $d_{max}(w_1,w_2,\dots,w_m) \leq MinDist[k]$. The search continues until k sets of data points have been added to A.

The following theorem shows the correctness of the hierarchical algorithm.

Theorem 1. *Let A be the answer set returned by $GTP-HA$. A includes k sets of data points that have k smallest total travel distances with respect to S and D of a group.*

Proof. (By contradiction) Assume that a set of data points, $\{p_1^j,p_2^j,\dots,p_m^j\}$, have j^{th} $(1 \leq j \leq k)$ minimum total travel distance with respect to S and D of a group. The set of data points $\{p_1^j,p_2^j,\dots,p_m^j\}$, may not have been included in A for two reasons: (i) the set of data points $\{p_1^j,p_2^j,\dots,p_m^j\}$ or R^*-tree nodes $\{w_1,w_2,\dots,w_m\}$ containing the set of data points $\{p_1^j,p_2^j,\dots,p_m^j\}$ have been pruned or (ii) the algorithm has terminated before the set of data points $\{p_1^j,p_2^j,\dots,p_m^j\}$ have been included in A.

We know that the total travel distance of a set $\{p_1^j,p_2^j,\dots,p_m^j\}$ is within $d_{min}(w_1,w_2,\dots,w_m)$ and $d_{max}(w_1,w_2,\dots,w_m)$, where $w_1,w_2,\dots,w_m$ represent $p_1^j,p_2^j,\dots,p_m^j$ or a minimum bounding rectangle of R^*-tree node containing

$p_1^j, p_2^j, \ldots, p_m^j$, respectively. In the algorithm, $Mindist[k]$ represents the current k^{th} smallest distance of already computed $d_{max}(w_1, w_2, \ldots, w_m)$s and remains same or decreases with the execution of the algorithm. According to the assumption, $d_{min}(w_1, w_2, \ldots, w_m) \leq Mindist[k]$. However, in the algorithm, $(w_1, w_2, \ldots, w_m)$ is only pruned if $d_{min}(w_1, w_2, \ldots, w_m) > Mindist[k]$, which contradicts the assumption.

Further, the algorithm terminates when k sets of data points have been dequeued from Q_p, where elements of Q_p are maintained in order of d_{min}s. If $\{p_1^j, p_2^j, \ldots, p_m^j\}$ is not included in A then according to our algorithm $\{p_1^j, p_2^j, \ldots, p_m^j\}$ has not been dequeued as one of k sets of data points and has total travel distance larger than the k^{th} smallest total travel distance, which again contradicts the assumption. □

To further improve the pruning capabilities of our hierarchical algorithm, we can first determine the upper bound of the k^{th} minimum total travel distance for the group trip and use it to prune data points/R^*-tree nodes while computing the optimal total travel distance. We use the following heuristics to determine the upper bound of the k^{th} minimum total travel distance for the group trip.

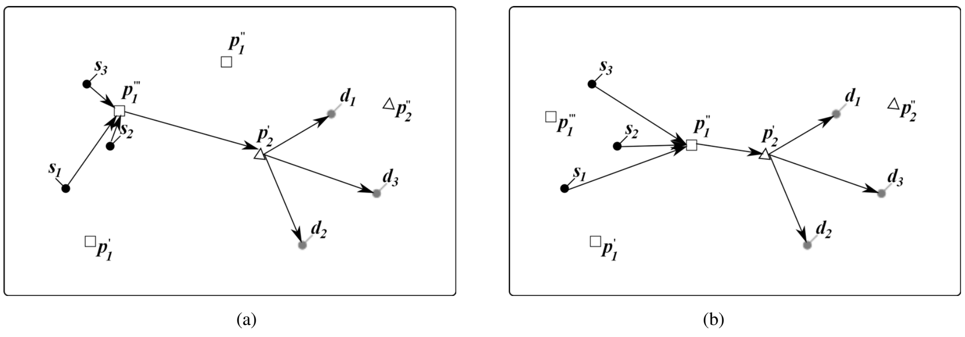


Fig. 2. Two example scenarios (a) and (b) for $k = 1$ and $m = 2$, where Heuristics 1 and Heuristic 2 are applied, respectively, to compute the upper bound.

Heuristic 1 *Let p_1 be the GNN with respect to S, p_j be the nearest neighbor from p_{j-1} for $2 \geq j < m$, and p_m be the k^{th} GNN with respect to $p_{m-1} \cup D$. The upper bound of the k^{th} minimum total travel distance for the group trip is computed as $f(S, D, p_1, p_2, \ldots, p_m)$.*

Heuristic 2 *Let p_1 be the GNN with respect to $S \cup D$, p_j be the GNN with respect to $p_{j-1} \cup D$ for $2 \geq j < m$, and p_m be the k^{th} GNN with respect to $p_{m-1} \cup D$. The upper bound of the k^{th} minimum total travel distance for the group trip is computed as $f(S, D, p_1, p_2, \ldots, p_m)$.*

In Heuristic 1, the upper bound is computed based only on the distance, whereas in Heuristic 2, the direction of D from S in addition to the distance are used for upper bound computation. Depending on the distribution of locations of sources, destinations and data points, Heuristic 1 or Heuristic 2 can provide the smaller upper bound of the k^{th} total travel distance. Figure 2 shows example scenarios for $k = 1$ and $m = 2$, where Heuristics 1 and Heuristic 2 provide smaller upper bound in Figure 2(a) and 2(b), respectively. In Figure 2(a), the upper bound is computed for the pair (p_1''', p_2') using Heuristics 1 and in Figure 2(b), the upper bound is computed for the pair (p_1'', p_2') using Heuristics 2.

Algorithm 4: UpperBound_MinDist(S, D, k)

```
     Input  : $S = \{s_1, s_2, \ldots, s_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $k$.
     Output: The upper bound of the $k^{th}$ minimum total travel distance.
     // Heuristic 1
4.1  $p_1 \leftarrow FindGNN(1, S)$
4.2  $i \leftarrow 2$
4.3  for $i < m$ do
4.4      $p_i \leftarrow FindNN(1, p_{i-1})$
4.5  $p_m \leftarrow FindGNN(k, p_{m-1} \cup D)$
4.6  $Dist_1 \leftarrow f(S, D, p_1, p_2, \ldots, p_m)$
     // Heuristic 2
4.7  $p_1 \leftarrow FindGNN(1, S \cup D)$
4.8  $i \leftarrow 2$
4.9  for $i < m$ do
4.10     $p_i \leftarrow FindGNN(1, p_{i-1} \cup D)$
4.11 $p_m \leftarrow FindGNN(k, p_{m-1} \cup D)$
4.12 $Dist_2 \leftarrow f(S, D, p_1, p_2, \ldots, p_m)$
4.13 if $Dist_1 \leq Dist_2$ then
4.14     return $Dist_1$
4.15 else
4.16     return $Dist_2$
```

Algorithm 4 shows the steps for computing the upper bound of the k^{th} minimum total travel distance for the group trip. The algorithm determines the upper bound using Heuristic 1 and Heuristic 2 separately (Line 4.3 and Line 4.6, respectively) and returns the smaller upper bound of the k^{th} total travel distance. Note that the function *FindGNN* and *FindNN* used in Algorithm 4 to compute the GNN and the nearest neighbor is the same function used in Algorithm 2.

After computing the upper bound, in Algorithm 3, $MinDist[k]$ is initialized with the upper bound of the k^{th} minimum total travel distance for the group trip instead of ∞ (Line 3.3) and used to prune data points/R^*-tree nodes using Lemma 2 (Line 3.17).

5 Experiments

In this section, we evaluate the performance of our proposed algorithm *k*GTP-HA with the base line algorithm *k*GTP-IA through an extensive set of experiments. In our experiments, we use two variants of our hierarchical algorithms: one without upper-bound of

k^{th} minimum total travel distance, called *k*GTP-HA1, and the other with pre-computed upper-bound, called *k*GTP-HA2. We use both real and synthetic data sets in our experiments. The real data set C is taken from 62,556 points of interests (i.e., data points) of California. We generate synthetic data sets using a uniform (U) and a Zipfian (Z) distribution, respectively. We vary the size of U and Z as 5000, 10,000, 15,000, and 20,000 point locations. For all data sets, the data space is normalized into a span of $10,000 \times 10,000$ square units. Since we need m categories of data points, we equally divide the set of data points of a dataset and index them using R*-trees. We run the experiments on a desktop with a Intel Core 2 Duo 2.40 GHz CPU and 4 GBytes RAM.

In realistic scenarios where a group of users plans a trip that involves different types of data objects, the number of different data points is typically limited to 2 or 3. A group may want to go to dinner, enjoy a movie at a cinema afterwards, and possibly go later to a pub, but it is unlikely that a group trip will be planned in advance for a larger number of places, i.e., data points. Thus we set the number of data types (m) to 2 and 3 in our experiments. A group will usually know in advance in which order they will visit the different places. This also motivates to consider ordered *k*GTP queries in our experiments. Note that our algorithm works with any number of data types for both ordered and flexible *k*GTP queries.

We vary different parameters: the group size (n), 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. In all experiments, we measure IO costs and the query processing time to measure the efficiency of our algorithms. Table 2 summarizes the values used for each parameter in our experiments and their default values.

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

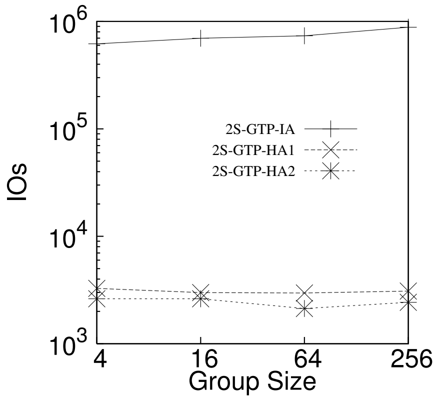
Table 2. Experiment Setup

We first present the experimental results for two types of data points (i.e., $m = 2$), which we call a 2 stops GTP (2S-*k*GTP) query. Then we present the results for three types of data points (i.e., $m = 3$), which we call a 3 stops GTP (3S-*k*GTP) query.

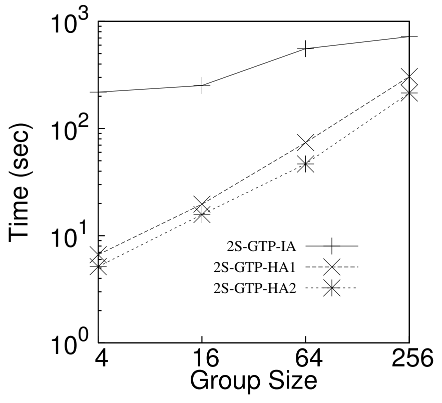
5.1 2S-*k*GTP Queries

In this section, for each set of experiments, we execute 100 2S-*k*GTP queries and show the average results. These queries are generated randomly in the total space.

Effect of Group Size Figures 3(a) and 3(b) show the IOs and the processing time, respectively, required by 2S-GTP-IA, 2S-GTP-HA1, and 2S-GTP-HA2 for different group sizes. Both IOs and the processing time increase with the increase of group size for all three approaches. Figure 3(a) shows that 2S-GTP-IA requires on average two order of magnitude more IOs than that of both 2S-GTP-HA1 and 2S-GTP-HA2. We observe in Figure 3(b) that the processing time of 2S-GTP-IA is on average an order of magnitude higher than that of 2S-GTP-HA1. We also observe that 2S-GTP-HA2 takes on average 20% less IOs and 27% less processing time than 2S-GTP-HA1, which is expected due to a tighter upper-bound of the termination condition of 2S-GTP-HA2.

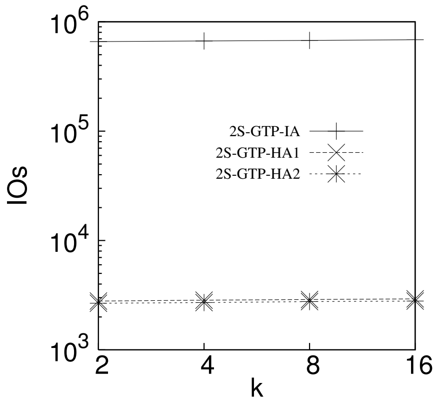


(a)

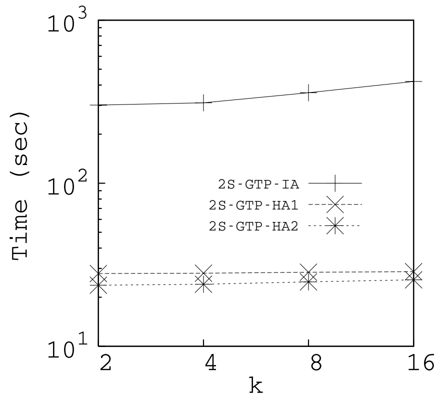


(b)

Fig. 3. Effect of group size (data set *C*)



(a)



(b)

Fig. 4. Effect of *k* (data set *C*)

Effect of *k* In this set of experiments, we vary the value of *k* as 2, 4, 8, and 16. In Figure 4(a), we observe that the IO cost slightly increases with the increases of *k* for 2S-GTP-IA and remains almost constant for 2S-GTP-HA1 and 2S-GTP-HA2. From the experimental results, we find that 2S-GTP-IA requires at least two orders of magnitude times more IOs than that of both 2S-GTP-HA1 and 2S-GTP-HA2, and 2S-GTP-HA2 takes on average 4% less IOs than that of 2S-GTP-HA1.

Figure 4(b) shows that the processing time of 2S-GTP-IA is at least one order of magnitude higher than that of both 2S-GTP-HA1 and 2S-GTP-HA2. On the other hand, the processing time of 2S-GTP-HA2 is on average 13% lower than that of 2S-GTP-HA1, which is expected as 2S-GTP-HA2 uses a tighter upper-bound to facilitate more pruning capabilities.

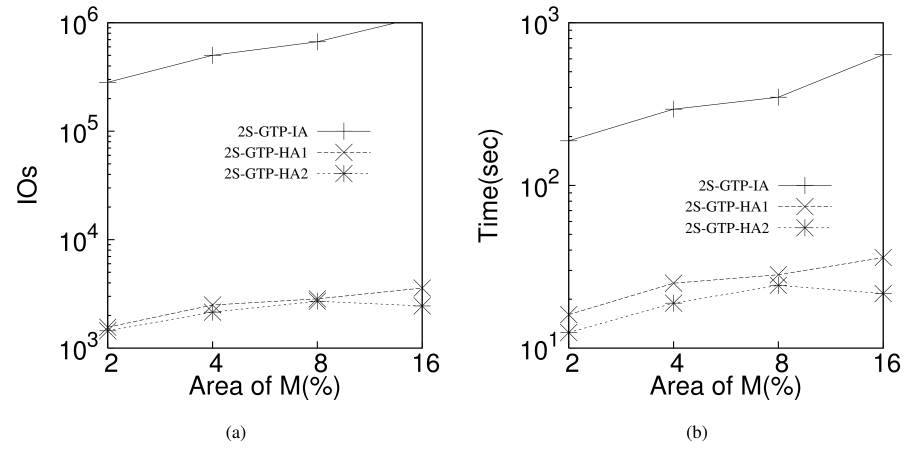


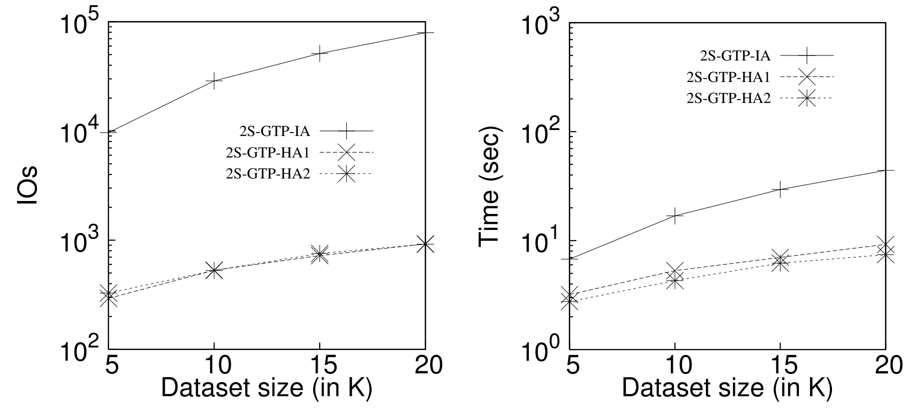
Fig. 5. Effect of query area M(data set C)

Effect of Query Area (M) In this set of experiments, we vary the query area M as 2%, 4%, 8%, and 16% of the data space. Figures 5(a) and 5(b) show the IO cost and the processing time, respectively, required by 2S-GTP-IA, 2S-GTP-HA1, and 2S-GTP-HA2 for different M. The IO cost and the processing time increase with the increase of M area for all three algorithms as for a larger M we need retrieve to access more data points from R*-trees than that of a smaller M.

Figure 5(a) shows that 2S-GTP-IA requires at least two orders of magnitude more IOs than that of both 2S-GTP-HA1 and 2S-GTP-HA2, and 2S-GTP-HA2 takes 14% less IOs than that of 2S-GTP-HA. Similarly, Figure 5(b) shows that the processing time of 2S-GTP-IA is on average one order of magnitude higher than that of both 2S-GTP-HA1 and 2S-GTP-HA2, and 2S-GTP-HA2 takes on average 25% less processing time than that of 2S-GTP-HA1.

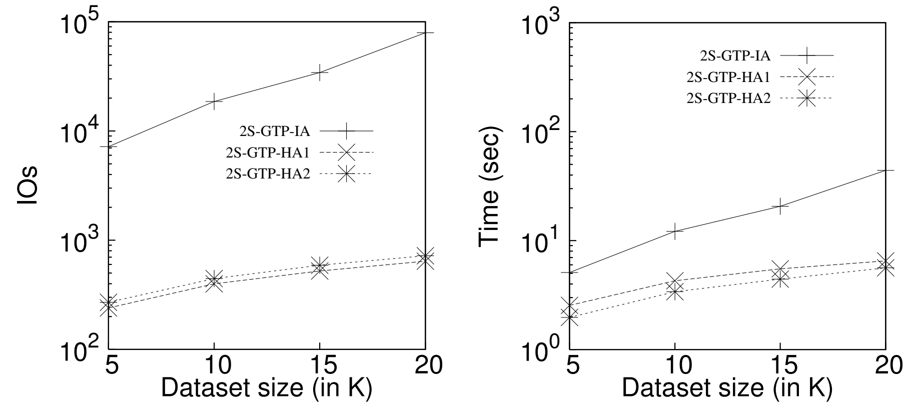
Effect of Data Set Size In this set of experiments, we vary the dataset size as 5000, 10000, 15000, and 20000 for both uniform (U) and Zipfian (Z) distributions.

Figures 6(a) and 6(b) show the IO cost and processing time, respectively, for different data set sizes with U distribution. The experimental results show that both 2S-GTP-HA1 and 2S-GTP-HA2 outperform 2S-GTP-IA in a greater margin for a larger dataset in terms of both IOs and processing time. Figures 7(a) and 7(b) show the IO



(a) (b)

Fig. 6. Effect of dataset size (dataset U)



(a) (b)

Fig. 7. Effect of dataset size (dataset Z)

cost and processing time, respectively, for different data set sizes with Z distribution and we observe that the experimental results for Z distribution follow similar trends to U distribution.

5.2 3S-*k*GTP Queries

From the experimental results of 2S-*k*GTP queries, we observe that both of our hierarchical approaches significantly outperform the iterative approach. Thus, in the next set of experiments for 3S-*k*GTP queries, we only evaluate the performance of two variants of our hierarchical approaches, 3S-GTP-HA1 and 3S-GTP-HA2. In these experiments, we vary different parameters such as group size, k, M, and dataset size. Figures 8(a)

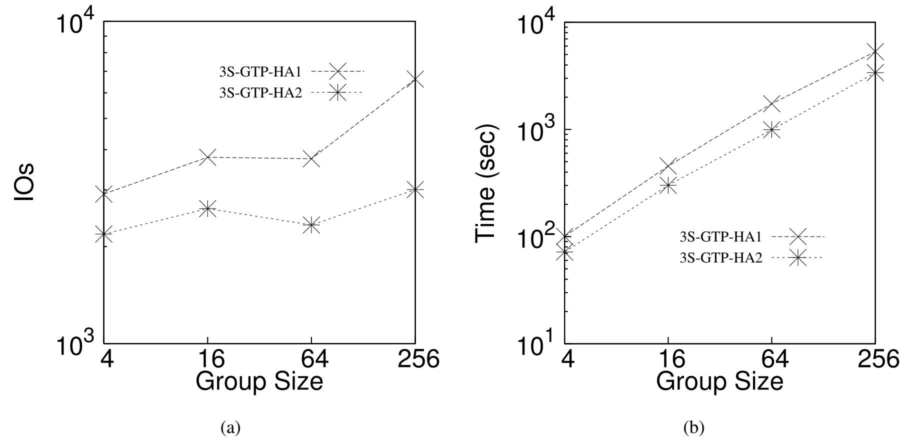


Fig. 8. Effect of group size (data set *C*)

and 8(b) show the effect of varying group size on IO cost and processing time, respectively, for California dataset. The experimental results show that 3S-GTP-HA2 always outperforms 3S-GTP-HA1 in terms of both IOs and processing time for any group size. We observe in Figures 8(a) that the processing time of 3S-GTP-HA2 is on average 55% less than that of 3S-GTP-HA1. Similarly, 3S-GTP-HA2 requires on average 64% less IOs than that of 3S-GTP-HA1. The results also show that 3S-GTP-HA2 outperforms 3S-GTP-HA1 in a greater margin for a larger group size.

We omit the experimental results for other parameters (e.g., k, M, and dataset sizes) for space constraint. The results show similar behavior as of 2S-GTP queries evaluation presented in the previous section. However, the processing time and IOs of 3S-GTP queries are higher than those of 2S-GTP queries, which is expected. Since in realistic scenarios, the number of different data points is typically limited to 2 or 3 for a kGTP query, the trend of increased processing overheads for a larger value of m would not effect the applicability of our algorithms.

6 Conclusion

In this paper, we introduced a new query type: the k group trip planning (kGTP) query. This query has many real world applications, in particular with the increased use of location-based social networks. We proposed an efficient hierarchical algorithm to evaluate kGTP queries. Our hierarchical algorithm evaluates the query with a single search on the database. We also developed an iterative approach as baseline method, which transforms kGTP queries into group nearest neighbor queries. We performed extensive experiments to show the efficiency of our algorithms and benchmark our hierarchical approach against the baseline method. The hierarchical algorithm performs on average an order of magnitude time faster and requires on average two orders of magnitude less IOs than the iterative approach. In the future, we plan to evaluate GTP queries in road networks. We also aim to protect the location privacy [21, 22] of users while evaluating

GTP queries, i.e., we will study scenarios where the group of users does not reveal their locations among each other.

Acknowledgments

This research has been done in the department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET). The work is supported by the research grant from BUET.

References

1. Facebook: `http://www.facebook.com`.
2. Google+: `http://plus.google.com`.
3. Loopt: `http://www.loopt.com`.
4. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE. (2004) 301
5. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. TODS **30**(2) (2005) 529–576
6. Deng, K., Sadiq, S.W., Zhou, X., Xu, H., Fung, G.P.C., Lu, Y.: On group nearest group query processing. IEEE TKDE **24**(2) (2012) 295–308
7. Li, Y., Li, F., Yi, K., Yao, B., Wang, M.: Flexible aggregate similarity search. In: SIGMOD. (2011) 1009–1020
8. Sharifzadeh, M., Kolahdouzan, M., Shahabi, C.: The optimal sequenced route query. The VLDB Journal **17**(4) (2008) 765–787
9. Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., Teng, S.H.: On trip planning queries in spatial databases. SSTD (2005) 273–290
10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD. (1984) 47–57
11. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. **19**(2) (1990) 322–331
12. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: International Symposium on Advances in Spatial Databases. (1995) 83–95
13. Li, F., Yao, B., Kumar, P.: Group enclosing queries. IEEE TKDE **23**(10) (2011) 1526–1540
14. Chen, H., Ku, W.S., Sun, M.T., Zimmermann, R.: The multi-rule partial sequenced route query. GIS (2008) 10:1–10:10
15. Ohsawa, Y., Htoo, H., Sonehara, N., Sakauchi, M.: Sequenced route query in road network distance based on incremental euclidean restriction. In: DEXA. (2012) 484–491
16. Malviya, N., Madden, S., Bhattacharya, A.: A continuous query system for dynamic route planning. In: ICDE. (2011) 792–803
17. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: ICDE. (2011) 900–911
18. Geisberger, R., Kobitzsch, M., Sanders, P.: Route planning with flexible objective functions. In: ALENEX. (2010) 124–137
19. Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie, X.: Searching trajectories by locations: an efficiency study. In: SIGMOD. (2010) 255–266
20. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD. (1995) 71–79
21. Hashem, T., Kulik, L., Zhang, R.: Privacy preserving group nearest neighbor queries. In: EDBT. (2010) 489–500
22. Mokbel, M.F., Chow, C.Y., Aref, W.G.: The new casper: query processing for location services without compromising privacy. In: VLDB. (2006) 763–774