Dynamic Group Trip Planning Queries in Spatial Databases

Anika Tabassum[1], Sukarna Barua[2], Tanzima Hashem[3] and Tasmin Chowdhury[4]
Dept of CSE, Bangladesh University of Engineering and Technology, Bangladesh
{sukarnabarua,tanzimahashem@cse.buet.ac.bd}{anika.tabassumcse41,tasmin.buet@gmail.com}

ABSTRACT

In this paper, we introduce the concept of "dynamic groups" for Group Trip Planning (GTP) queries and propose a novel query type Dynamic Group Trip Planning (DGTP) queries. The traditional GTP query assumes that the group members remain static or fixed during the trip, whereas in the proposed DGTP queries, the group changes dynamically over the duration of a trip where members can leave or join the group at any point of interest (POI) such as a shopping center, a restaurant or a movie theater. The changes of members in a group can be either predetermined (i.e., group changes are known before the trip is planned) or in real-time (changes happen during the trip). In this paper, we provide efficient solutions for processing DGTP queries in the Euclidean space. A comprehensive experimental study using real and synthetic datasets shows that our efficient approach can compute DGTP query solutions within few seconds and significantly outperforms a naive approach in terms of query processing time and I/O access.

CCS CONCEPTS

•**Information systems** →*Database query processing;*

KEYWORDS

Group Trip Planning Query, Dynamic Group, Spatial Database

ACM Reference format:
Anika Tabassum[1], Sukarna Barua[2], Tanzima Hashem[3] and Tasmin Chowdhury[4] . 2017. Dynamic Group Trip Planning Queries in Spatial Databases. In *Proceedings of SSDBM '17, Chicago, IL, USA, June 27-29, 2017,* 6 pages.
DOI: 10.1145/3085504.3085584

1 INTRODUCTION

A Group Trip Planning (GTP) query [2, 3] assumes that the group remains static during the trip, i.e., starting from the dispersed source locations, the group members travel together via a sequenced set of points of interests (POIs) such as a shopping center, a restaurant and a movie theater, and finally head towards their different destination locations. The strict constraint of visiting the whole trip as a static group poses serious limitations to its applicability in many practical scenarios. For example, some members may want to join the group in the middle of a trip instead of the first POI. Some other members may want to leave the trip earlier than from the last POI type. The join and/or leave events of group members can be preplanned (i.e., known before the start of the group trip) or may happen in real time. To address this, we introduce a *Dynamic Group Trip Planning (DGTP) query* that can handle the dynamic group behavior and computes a group trip with the minimum travel distance.

Straightforward application of existing GTP algorithms [2, 3] for every change of group members in a DGTP query would incur extremely high processing overhead due to the retrieval of same POIs multiple times and repeated computations. In this paper, we develop an efficient approach for processing DGTP queries for both Euclidean space. Our approach can handle both predetermined and real time changes of members in the group; users can join, leave or rejoin anytime during a group trip.

We develop novel techniques to refine the POI search region. Using elliptical properties, we prove that any POI outside the search region cannot minimize the total trip distance of group members. In addition, we extend the dynamic programming approach [2] for GTP queries to efficiently compute the aggregate trip distance by considering the dynamic group behavior and re-evaluations of answers for a DGTP query. Extensive experiments using real and synthetic datasets show that our approach significantly outperforms a naive approach in terms of I/O access and processing time. In summary, our contributions are as follows.

(i) We introduce the concept of dynamic groups, and DGTP queries for Euclidean space.

(ii) We propose an efficient approach to process DGTP queries. Specifically, we develop techniques to refine the POI search space, and thereby reduce the number of POI retrieval from the database. In addition, we develop a dynamic programming algorithm to efficiently compute optimal group trips based on the retrieved POIs from the database.

(iii) We conduct extensive experiments using real data sets in the Euclidean space to show the effectiveness of our efficient approach over a naive approach.

2 RELATED WORK

Efficient computation of trips for a single user [5, 6] as well as for groups [2, 3] have been studied in both Euclidean space and road networks. Group trip planning (GTP) has been introduced in [3] for the Euclidean space. The proposed algorithm in [3] is not extensible for processing GTP queries in road networks. In [1, 2], the authors proposed efficient GTP algorithms for road networks. Recently, in [4], the authors proposed a new variant of a GTP query, subgroup trip planning queries, that return the optimal trips for different subgroup size. These approaches for processing GTP queries and variants assume that all group members visit every POI type. On the other hand, we overcome this limitation; in a DGTP query, a group member can visit any number of POI types.

In [2], the authors refine the POI search region as elliptical regions, where the elliptical property ensures that the distance between two foci of the ellipse via a POI outside the ellipse is greater than or equal to the length of the major axis. The authors developed techniques to determine the foci of the ellipses, and set the length of the major axis such that a POI outside the ellipse cannot

SSDBM '17, Chicago, IL, USA

DOI: 10.1145/3085504.3085584

become part of an optimal trip. However, these techniques are not applicable in our scenario as in our approach, members can join and/or leave a group at any POI type.

3 PROBLEM FORMULATION

A DGTP query enables a dynamic group to plan a group trip with the minimum aggregate travel distance.The POI types and the order of visiting POI types (e.g., restaurant before a shopping center) are specified before a group trip is planned. In a DGTP query, it is essential that in addition to the POI types, the order of visiting POI types is fixed before the start of the group trip because group members decide their trips based on the order of visiting POI types. A group member may decide to join a group trip if a shopping center and a movie theater are visited in a sequence. A group member's trip with respect to a group trip can be formally defined as a follows:

Definition 3.1. **(A Trip $T_i = \{s_i, PT_i, d_i\}$).** Given a set of m ordered POI types $PT = \{t_1, t_2, \ldots, t_m\}$ for a group trip, a trip T_i of user u_i starts from source location s_i, goes through a set of POI types PT_i that is a contiguous subset of PT, and ends at destination d_i.

A user may participate in a group trip multiple times. Consider an example, where $PT = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. A user u_i first participates in POI types t_1 and t_2 with respect to her source and destination locations s_i^1 and d_i^1, and then again participates in POI types t_4 and t_5 with respect to her source and destination locations s_i^2 and d_i^2. Thus, in such a scenario, a DGTP query considers multiple trips of a group member while planning a group trip. We denote multiple trips of user u_i as $T_i^1, T_i^2, \ldots, T_i^{k_i}$, where u_i participates in k_i trips and $T_i^j = \{s_i^j, PT_i^j, d_i^j\}$.

Depending upon the source and destination locations, and POI types of the trips of group members, before the start of a group trip, a DGTP query identifies the locations of POIs, one from every type in $PT = \{t_1, t_2, \ldots, t_m\}$, that minimize the group trip distance. Before the start of a group trip, the group trip distance is the summation of the individual trip distances, where a trip distance is measured as the travel distance between the source and destination locations via the POIs of types specified in a trip. We use function $dist(p, q)$ to compute the Euclidean distance between two spatial points p and q.

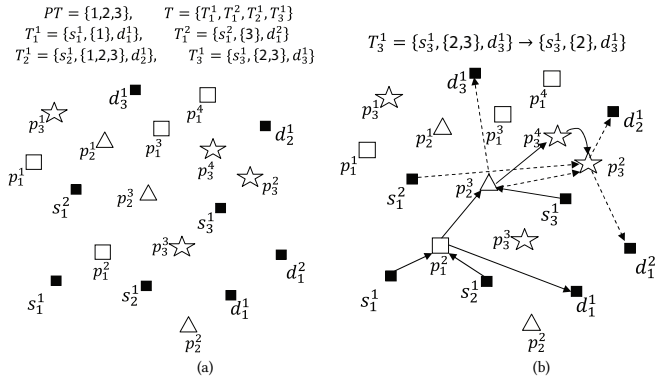

Figure 1: A DGTP query scenario

When an existing trip changes or a new trip is added in real-time at any POI, the initially optimal set of POIs retrieved by a DGTP query may no longer minimize the group trip distance for the remaining part of the group trip. To address this scenario, a DGTP query recomputes and identifies the POI locations for the not visited POI types in $PT = \{t_1, t_2, \ldots, t_m\}$ that minimize the group trip distance for the remaining part of the group trip.

Thus, the evaluation of a DGTP query is performed in two phases. Initially, based on initial trips of group members, a DGTP query returns an optimal set of POI locations, one for each type in $PT = \{t_1, t_2, \ldots, t_m\}$. After any change in an existing trip or joining of a new member with new trips, a DGTP query recomputes the location of optimal POIs for not visited POI types in PT. Formally, a DGTP query is defined as follows:

Definition 3.2. **(A DGTP Query).** Given a set of locations of POIs L in a two dimensional space, a set of ordered POI types $PT = \{t_1, t_2, \ldots, t_m\}$, and a set of trips $T = \{\bigcup_{i \in n} \bigcup_{1 \le j \le k_i} T_i^j\}$ of n group members where k_i is the number of trips specified by user u_i, a DGTP query initially returns $P_0 = \{p_{t_1}, p_{t_2}, \ldots, p_{t_m}\}$, where $P_0 \subseteq L$ and p_i represents a POI of type $t_i \in PT$, such that the group trip distance is minimized. Let a change in T occurs when the group visits POI p_{t_j}. A DGTP query returns $P_j = \{p_{t_j}, p_{t_{j+1}}, \ldots, p_{t_m}\}$, where $P_j \subseteq L$ and p_i represents a POI of type $t_i \in PT$, such that the group trip for the remaining part of the group trip is minimized.

Example. Figure 1(a) shows a DGTP query scenario, source, and destination locations of three users, and sample POI locations of three POI types $PT = \{1, 2, 3\}$. In the figure, rectangles represent type 1 POI, triangles represent type 2 POI, stars represent type 3 POI, and p_i^j denotes the j-th POI of type i. For this example, initially user 1 specifies two informed trips $T_1^1 = \{s_1^1, \{1\}, d_1^1\}$ and $T_1^2 = \{s_1^2, \{3\}, d_1^2\}$, user 2 specifies one informed trip $T_2^1 = \{s_2^1, \{1, 2, 3\}, d_2^1\}$, and user 3 specifies one informed trip $T_3^1 = \{s_3^1, \{2, 3\}, d_3^1\}$. Thus the trip set T contains four trips where $T = \{T_1^1, T_1^2, T_2^1, T_3^1\}$. Let a possible answer set for this example be $P_0 = \{p_1^2, p_2^3, p_3^4\}$. Figure 1(b) shows that when group trip is at second POI p_2^3, user 3 changes its trip from $T_3^1 = \{s_3^1, \{2, 3\}, d_3^1\}$ to $T_3^1 = \{s_3^1, \{2\}, d_3^1\}$, i.e., user 3 leaves immediately after visiting type 2 POI (the dashed line from p_2^3 to d_3^1). As trip set T changes, the optimal answer requires a re-computation for the remaining POI types to be visited (type 3). The figure shows that a possible recomputed answer could be $P_2 = \{p_3^2\}$. The solid line from p_2^3 to p_3^4 is the initial optimal path while dashed line from p_2^3 to p_3^2 and other dashed lines comprise the recomputed optimal path for the group.

4 A NAIVE APPROACH: NAIVEDGTP

We first develop a naive approach *NaiveDGTP* for DGTP queries. Our naive approach works in two phases: The initial phase retrieves all POIs from the database and computes an optimal answer for the initial trips, and the second phase is an update operation that updates existing optimal answer with respect to trip changes or addition of new trips in real time. The algorithm is briefly explained below where Step 1 comprises the initial phase and Step 2 describes the update phase.

Step 1 - We retrieve all POIs of required types from the database. Then we apply a dynamic programming (dp) algorithm that

Table 1: Columns of dp table with respect to DGTP scenario shown in Fig. 1(a). Each cell in a column stores a POI p_k^i and aggregate sum v_k^i.

Column 1	*Column* 2	*Column* 3
$< p_1^1, v_1^1 >$	$< p_2^1, v_2^1 >$	$< p_3^1, v_3^1 >$
$< p_1^2, v_1^2 >$	$< p_2^2, v_2^2 >$	$< p_3^2, v_3^2 >$
$< p_1^3, v_1^3 >$	$< p_2^3, v_2^3 >$	$< p_3^3, v_3^3 >$
$< p_1^4, v_1^4 >$		$< p_4^3, v_3^4 >$

computes an optimal answer from the retrieved set of POIs based on set of trips T and set of ordered POI types PT. The algorithm uses a table like data structure (shown in Table 1) consisting of m columns of cells. Each cell c_k^i in column k stores two things: a POI p_k^i of type t_k and partial aggregate distance v_k^i up to that POI. we compute the partial aggregate sum v_k^i for each cell c_k^i as follows.

(a) For $k = 1$ (column 1), we compute the sum of distances from (i) sources to p_1^i for trips that join at type t_1 and (ii) destination to p_1^i for trips that leave from type t_1. In this way, for all i v_1^i is computed and stored in c_1^i.

(b) For $2 \leq k \leq m$, from each cell c_{k-1}^j of the previous column $k-1$, we compute a sum of (i) the partial sum stored at c_{k-1}^j, (ii) the distance from p_{k-1}^j to p_k^i multiplied by the number of users who continue the trip from t_{k-1} to t_k, (iii) the distances from sources to p_k^i for trips that join at type t_k, and (iv) the distances from destinations to p_k^i for trips that leave from type t_k. Then the minimum of all such sums (found for all js) is calculated and stored at c_k^i. In this way, v_k^i is computed for all i and stored in c_k^i. After this answer set is computed, the group starts visiting the optimal POIs.

Step 2 - This step is performed whenever some new trips are added or trip changes occur in real-time. Based on the changes, POI type set PT is updated by removing POI types $t_1, t_2, \ldots, t_k$ that have already been visited. Thus, the modified POI type set becomes $PT' = \{t_{k+1}, t_{k+2}, \ldots, t_m\}$. The updated trip set T' is found as follows: (i) all new trips are added and trip changes are incorporated, (ii) all trips T_i^js in T which are completely unvisited are kept as is, (iii) all trips T_i^js which have been completely visited are removed from T, (iv) For trips T_i^js which are partially visited, the unvisited parts of the trip are kept, the visited part is removed from T_i^j. For these partial trips, the source s_i^j is also updated to be the current POI location p_k. After T and PT are updated, a new DGTP query is run over the modified T' and PT' to retrieve the updated optimal POI set.

NaiveDGTP runs the dp algorithm multiple times (once for initial phase and once for every update) on the entire data space. Hence it will result in extensive I/O overhead and processing time. In Section 5, we describe an efficient solution to process DGTP queries.

5 AN EFFICIENT APPROACH: FASTDGTP

In this section, we present an efficient approach *FastDGTP* for processing DGTP queries. *FastDGTP* has the following two key improvements over the *NaiveDGTP* approach: (i) It exploits the trip information of users to compute a pruning bound that allows it to search a small data space instead of the entire database, and (ii) It stores POIs in a cache and re-uses those POIs to avoid retrieving the same POIs again in different runs of the update phase.

The initial phase of the algorithm, the computation of the initial optimal answer set, has the following key steps:

Step 1 - The algorithm first computes a heuristic answer for the DGTP query.

Step 2 - Using the heuristic answer and trip set T, the algorithm computes three pruning bounds for refining the search region. The pruning bounds ensure that the algorithm searches only a limited set of POIs rather than the entire data space in the dp algorithm to compute the optimal answer. It significantly reduces the computational overhead incurred by the inefficient naive approach.

Step 3 - The algorithm performs a range query to retrieve all candidate POIs residing inside the refined search space.

Step 4 - The algorithm applies the dp algorithm (described in Step 1 of Section 4) to compute the optimal answer set from the candidate set of POIs retrieved by Step 3 above. In addition, the algorithm also saves the retrieved POIs and the computed pruning bounds to use in the next update phase.

The update phase, which is executed to update optimal answer when T changes, consists of the following steps:

Step 1 - The algorithm updates trip set T and POI set PT using the technique described in Step 2 of Section 4. This creates the modified T' and PT'.

Step 2 - The algorithm computes new pruning bounds for the search region based on existing optimal answer and updated trip information T' and PT'. Note that instead of using a heuristic answer, we use the exiting optimal answer to compute bounds in the update phase.

Step 3 - The algorithm executes a range query to retrieve candidate POIs residing inside the new bound. Unlike the range query executed in the initial phase, the range query in update phase retrieves only those POIs that are within the newly computed bound and at the same time lies outside the previous bound. As there will be significant overlap between bound regions of successive update phases, we do not retrieve those POIs that have already been retrieved by the previous update phase. Instead we reuse those POIs from the cache. This strategy will result in only a very small set of POIs to be retrieved during each update phase. This caching saves the computational overhead to access the R-tree than if we would have retrieved all POIs within the new bound.

Step 4 - The algorithm uses the same dp technique used earlier to compute the updated optimal answer set. The optimal answer returned by update phase contains one POI from each unvisited POI type in PT'. The algorithm also saves the computed bound and retrieved candidate POIs in a cache to use in the future update step.

The following sections describe the key components of *FastDGTP* approach in detail.

5.1 Computation of Heuristic Answer

In the initial phase of the *FastDGTP* approach, a heuristic answer is required to compute the initial bounds for the search region. Although any heuristic technique is applicable, we use the following greedy approach that selects m POIs from the database, one POI for each type in PT.

We find the POI p_1 of type t_1 from the database such that $\sum_{i,j,J(i,j,1)=1} dist(s_i^j, p_1) + \sum_{i,j,L(i,j,1)=1} dist(d_i^j, p_1)$ is minimum among all POIs of the first type. Then we find sequentially $m-1$ POIs $p_2, p_3, ..., p_m$ of the remaining $m-1$ types such that for each such POI p_k of type t_k, the sum of the following three terms is minimum among all POIs of the corresponding type: (i) $dist(p_{k-1}, p_k) * n'$ where n' is the number of trips that visit p_k from p_{k-1}, (ii) $\sum_{i,j,J(i,j,k)=1} dist(s_i^j, p_k)$, and (iii) $\sum_{i,j,L(i,j,k)=1} dist(d_i^j, p_k)$. So our heuristic does actually work by finding a POI of each type such that it is the Group Nearest Neighbor (GNN) of previously selected POI multiplied by number of trips that visit a POI type t_k from a POI type t_{k-1}, set of sources locations of trips that join at the POI, and set of destination locations of trips that leave from the POI. After finding this heuristic set of POIs, we compute the individual trip distances of each user in the answer set, and then find the aggregate travel distance of all users. These distances are used for the bound computation explained in the next Section.

5.2 Computation of Search Region Bounds

In this section, we present how the trip set T and a candidate answer P can be exploited together to compute bounds for the search region. In the initial phase, P is computed by the heuristic method whereas in the update phase, existing optimal answer is used as P. Let us define the following terms:

- TD_i^j: The travel distance of trip T_i^j (jth trip of user u_i) in candidate answer P.
- TD: The aggregate travel distance of all trips of all users in P, so $TD = \sum_i \sum_j TD_i^j$.
- $TDmin_i^j$: The lower bound of TD_i^j.
- $TDmax_i^j$: The upper bound for TD_i^j.
- $dist(s, t)$: The Euclidean distance between spatial points s and t.
- $V(i, j, k)$: An indicator function where $V(i, j, k) = 1$ if trip T_i^j visits POI type t_k in group trip (i.e., $t_k \in PT_i^j$), and 0 otherwise.

5.2.1 Computation of $TDmin_i^j$. By definition, $TDmin_i^j$ is the lower bound of TD_i^j, i.e., the minimum distance that u_i has to travel for its jth trip in any candidate answer including the optimal one also. For each trip T_i^j, we compute $TDmin_i^j$ as follows: First we retrieve the nearest POI p_k of each type t_k in PT_i^j such that $dist(s_i^j, p_k) + dist(d_i^j, p_k)$ is minimized. Then $TDmin_i^j$ is found as, $TDmin_i^j = max_{1 \le k \le |PT_i^j|}(dist(s_i^j, p_k) + dist(d_i^j, p_k))$.

5.2.2 Computation of $TDmax_i^j$. For each trip T_i^j, we compute $TDmax_i^j$ using the aggregate travel distance TD of the current candidate answer P and $TDmin_i^j$s of all trips as follows: $TDmax_i^j = TD - \sum_{k,l,(k,l)\neq(i,j)} TDmin_k^l$. This is the upper bound of TD_i^j in optimal trip, i.e., T_i^j cannot be greater than $TDmax_i^j$; otherwise optimal trip aggregate distance would be greater than TD which is not possible.

5.2.3 Computation of elliptical regions. To compute the bounds for search region, we compute three different sets of ellipses/regions as follows: (i) For each trip $T_i^j \in T$, we form an ellipse E_i^j where each E_i^j has two foci at s_i^j and d_i^j, and has major axis equal to $TDmax_i^j$.

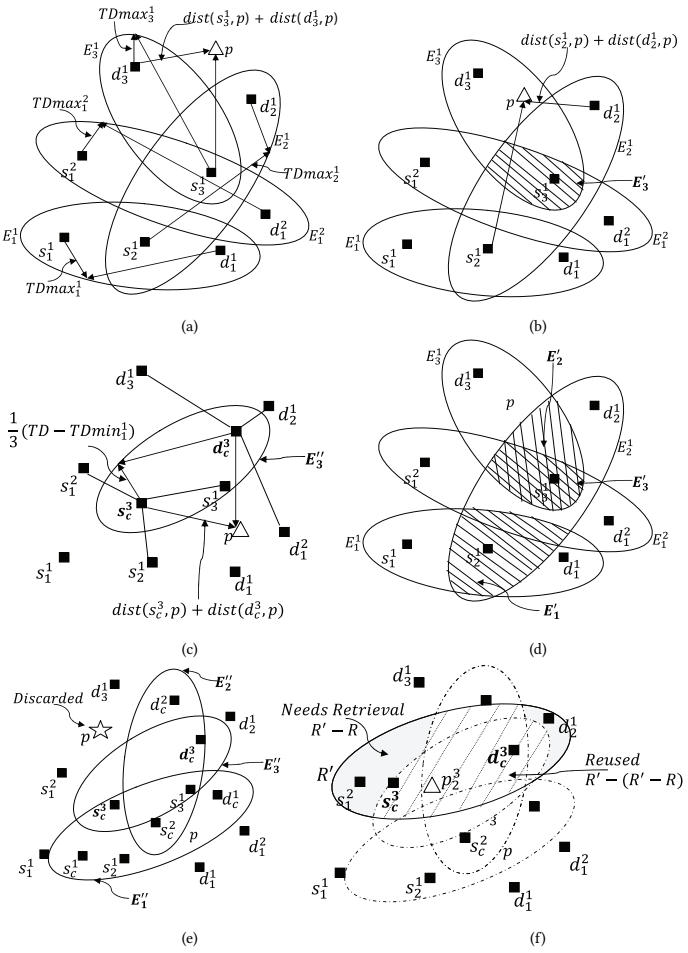

Figure 2: Computation of search region bounds.

In this way, we get $|T|$ ellipses. (ii) For all POI types in PT, we form m regions $E'_1, E'_2, ..., E'_m$, where each $E'_k = \cap_{i,j,V(i,j,k)=1} E_i^j$ represents a region for type t_k. (iii) For all POI types in PT, we form m ellipses $E''_1, E''_2, ..., E''_m$ as follows. Let n_k be the number of trips that visit POI type t_k, and s_c^k and d_c^k be the centroids of all source and destination locations of those n_k trips. Thus, $n_k = \sum_{i,j} V(i, j, k)$, $s_c^k = \frac{1}{n_k} \sum_{i,j,V(i,j,k)=1} s_i^j$, and $d_c^k = \frac{1}{n_k} \sum_{i,j,V(i,j,k)=1} d_i^j$. Then ellipse E''_k is an ellipse having foci at s_c^k and d_c^k, and major axis equal to $\frac{1}{n_k}(TD - \sum_{i,j,V(i,j,k)=0} TDmin_i^j)$.

Example. Fig. 2(a) shows the four ellipses E_1^1, E_1^2, E_2^1, E_3^1 with respect to the four trips T_1^1, T_1^2, T_2^1, and T_3^1. Search region bound R_1 is the union of all these ellipses. Fig. 2(b) shows the elliptical region E'_3 for type 3 where by definition $E'_3 = E_1^2 \cap E_2^1 \cap E_3^1$ since trips T_1^2, T_2^1, and T_3^1 visit type 3 POI. Fig. 2(d) shows all three regions E'_1, E'_2, E'_3 with respect to three POI types. Fig. 2(c) shows the ellipse E''_3 for type 3. For this ellipse, $n_3 = 3$ since three trips (T_1^2, T_2^1, and T_3^1) visit type 3, centroids $s_c^3 = \frac{1}{3}(s_1^2 + s_2^1 + s_3^1)$ and $d_c^3 = \frac{1}{3}(d_1^2 + d_2^1 + d_3^1)$ and major axis length is $\frac{1}{3}(TD - TD_1^1)$ since only trip T_1^1 does not visit type 3. Fig. 2(e) shows all three ellipses E''_1, E''_2, E''_3.

5.2.4 Search Region Bound 1.

Theorem 5.1. *The union of E_i^j's form a bound region R_1 for all trips combined. Any POI outside $R_1 = \cup_{i,j} E_i^j$ can be safely discarded, because it cannot provide a better solution than the current candidate answer.*

Proof. (By contradiction) Assume that p is a POI situated outside the region R_1, and visiting p provides a better solution than the current answer. Without loss of generality, assume jth trip T_i^j of user u_i visits this POI. Since p is outside the union region R_1, it is also outside of E_i^j. By the elliptical property, for any POI p outside of the ellipse E_i^j, $dist(s_i^j, p) + dist(d_i^j, p) > TDmax_i^j$, i.e., the major axis of E_i^j (see Fig. 2(a) where p is shown to be outside E_3^1 and $dist(s_3^1, p) + dist(d_3^1, p) > TDmax_3^1$). However, distance of trip T_3^1 cannot be greater the its upper bound $TDmax_3^1$. Thus, p cannot provide a better solution. *Contradiction.* □

Hence, we can prune our search region to R_1 as POIs outside R_1 can't give us a better solution than the current candidate answer. In Fig. 2(a), R_1 is the union of the four ellipses shown.

5.2.5 Search Region Bound 2.

Lemma 5.2. *Any POI of type t_k outside E'_k can be safely discarded as this cannot provide a better solution than the current candidate answer.*

Proof. (By contradiction) Assume that there is a POI p of type t_k that is outside the region E'_k, and visiting p provides a better solution than the current candidate answer. Since p is outside of E'_k (an intersection of ellipses), it is outside of at least one ellipse E_i^j, for which $V(i, j, k) = 1$. Now, $V(i, j, k) = 1$ implies trip T_i^j visits p, and p is outside of E_i^j. However, for any POI outside E_i^j $dist(s_i^j, p) + dist(d_i^j, p) >$ the length of the major axis of E_i^j, the upper bound for the corresponding trip. Thus, it cannot provide a better solution. *Contradiction.* □

We have proved that for each POI type t_k, we can form a bounding region which is the intersection of all ellipses E_i^j such that trip T_i^j visits that type. Any POI of type t_k outside this region can be discarded safely. The following theorem shows how we can form a combined search region for all types.

Theorem 5.3. *The union of E'_ks will form a bounding region R_2 for all types combined. Any POI outside this region $R_2 = U_{1<=k<=m} E'_k$ can be safely discarded, because it cannot provide a better solution than the current candidate answer.*

Proof. (By contradiction) Assume that p is a POI of type t_k situated outside the region R_2, and visiting p provides a better solution than the current best one. Since p is outside the region R_2, it is also outside of E'_k. Thus, according to Lemma 5.2, p cannot provide a better solution. Contradiction. □

Fig. 2(d) shows the search region bound R_2 (union of the striped areas).

5.2.6 Search Region Bound 3.

Lemma 5.4. *Any POI of type t_k outside the ellipse E''_k can be safely discarded because visiting such a POI cannot provide a better solution than the current candidate answer.*

Proof. Assume that, there is a POI p of type t_k outside the region E''_k, and visiting p provides a better solution than the current candidate answer. Since p is outside the ellipse E''_k and n_k trips visit p, the aggregate travel distance of the n_k trips through p will be at least equal to $\sum_{i,j,V(i,j,k)=1} (dist(s_i^j, p) + dist(d_i^j, p))$. Using Lemma 5.3 of [2], we can show that $\sum_{i,j,V(i,j,k)=1} (dist(s_i^j, p) + dist(d_i^j, p))$
$\geq n_k * (dist(s_c^k, p) + dist(d_c^k, p))$.
However, using the major axis property of ellipse E''_k, the Euclidean distance through p from s_c and d_c is greater than the major axis. We get, $dist(s_c^k, p) + dist(d_c^k, p) > \frac{1}{n_k}(TD - \sum_{i,j,V(i,j,k)=0} TDmin_i^j)$.
So, $\sum_{i,j,V(i,j,k)=1} (dist(s_i^j, p) + dist(d_i^j, p))$
$> n_k * \frac{1}{n_k}(TD - \sum_{i,V(i,j,k)=0} TDmin_i^j)$
$= TD - \sum_{i,j,V(i,j,k)=0} TDmin_i^j$.
Rearranging terms, we get $\sum_{i,j,V(i,j,k)=1} (dist(s_i^j, p) + dist(d_i^j, p)) + \sum_{i,j,V(i,j,k)=0} TDmin_i^j > TD$. The first part of the left hand side of the inequality is the aggregate travel distance of all trips that visit p ($V(i, j, k) = 1$), and second part is the aggregate travel distance of all trips that does not visit p ($V(i, j, k) = 0$). So the left hand side is the minimum aggregate travel distance of all trips considering p as part of a solution and is greater than current aggregate travel distance TD. Hence p does not give us a better solution. Contradiction. □

The following theorem shows how we can form a combined bound region using above lemma.

Theorem 5.5. *The union of all ellipses E''_k form a combined bounding region R_3. Any POI outside this bounding region $R_3 = \cup_{1 \leq k \leq m} E''_k$ can be safely discarded, because it cannot provide a better solution than the current candidate answer.*

Proof. Assume that p is a POI of type t_k situated outside the region R_3, and visiting p provides a better solution than the current candidate answer. Since p is outside the region R_3, it is also outside of E''_k. Thus, according to Lemma 5.4, p cannot give us a better solution. □

Fig. 2(e) shows the search region bound R_3 which is the union of three ellipses E''_1, E''_2, and E''_3. The figure also shows a POI p is discarded because it is outside the search region.

5.2.7 Combining the three bounds R_1, R_2, and R_3.

We form a combined bounding region $R = R_1 \cap R_2 \cap R_3$. Any POI outside the region R can be safely discarded, because it is outside of at least one of the three regions, and therefore can be discarded according to Theorem 5.1, or Theorem 5.3, or Theorem 5.5. Thus, our *FastDGTP* algorithm retrieves only those POIs that are inside the region R, and uses those POIs to compute the optimal answer.

Example. Fig. 2(f) shows a scenario of POI retrieval and POI reuse by the algorithm in the update phase with respect to user 3's trip change at a POI p_2^3 (Scenario of Fig. 1(b)). The solid ellipse (R') represents the newly computed bound for the search region while the dashed ellipses comprise the previous bound. The shaded region ($R' - R$) is the region needs to be retrieved from database. The striped region is the region whose POIs are retrieved from the cache. Note that Fig. 2(f) shows only the search region bounds 3 (Section 5.2.6) for simplicity.

6 EXPERIMENTS

In this section, we evaluate the performance of our proposed *FastDGTP* to process DGTP queries in the Euclidean space.

Table 2: Parameter Settings

Parameters	Values	Default
Group size (n)	2, 4, 8, 16, 32	8
Number of POI types(m)	2, 3, 4, 5, 6	3
Query area (A)(in sq. units)	2500, 10000, 22500, 40000, 62500	10000
Data set size (d_s)(Number of POIs)	5000, 10000, 15000, 20000	–

Data sets. We used the real California road network data set[1]. The California data set contains 87635 POIs of 63 different types. Its road network has 21048 nodes and 21693 edges. Two types of synthetic data sets were generated using uniform and zipfian distribution for POIs. In our experiments, we normalized its data space into a $1000x1000$ sq. units area. We used an R-tree data structure to index POIs.

Parameters and experiment settings. We varied the following four parameters: (i) total number of users n, (ii) number of POI types m, (iii) the query area A, i.e., the minimum bounding rectangle covering the source and destination locations, and (iv) size of data set d_s. Table 2 summarizes the range of parameter settings and their default values as per our experiments. For each experiment, we ran 100 randomly generated DGTP queries and computed the average processing time and average I/O access. All experiments were run in a computer with core i5 3.2 GHz processor and 6 GB RAM.

6.1 Performance in Processing DGTP query

In this section, we present the comparative performance analyses of *FastDGTP* and *NaiveDGTP* algorithms to process DGTP queries. We do not compare our approaches with a baseline approach that applies GTP algorithms for processing a DGTP query because the most recent and efficient GTP algorithm [2] cannot handle the dynamic group behavior.

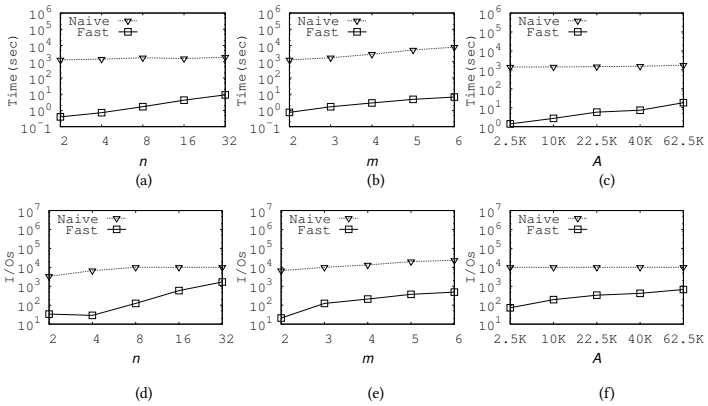

Figure 3: Effect of group size n, number of POI types m, and query area A (California data set).

Effect of n, m, A and d_s. Figure 3 shows the query processing time and I/O overhead incurred by the two approaches in California data set for different values of n, m and A. Figure 4 shows the same measures for different values of d_s in synthetic data sets. If either of n, m, A and size d_s increases, the processing time increases for both approaches. For all experiments, the I/O of *NaiveDGTP* is constant since it does not use any pruning bound, and uses entire data space to compute the optimal group trip. We estimated that *FastDGTP* takes on average approx. 2 orders of magnitude less time and 1 order of magnitude less I/O for n, 3 orders of magnitude less time and 3 orders of magnitude less I/O for m, 2 orders of magnitude less time and 1 order of magnitude less I/O for A and 2 orders of magnitude less processing time and 2 orders of magnitude less I/O for d_s than the naive approach.

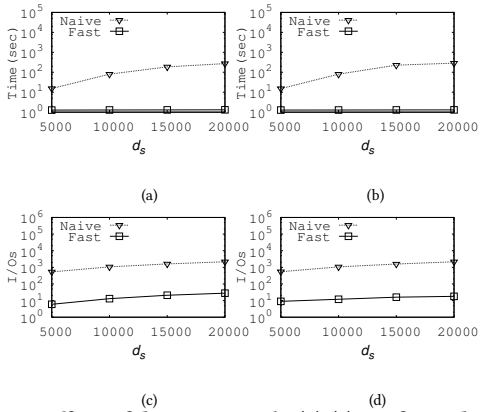

Figure 4: Effect of data set size d_s. (a),(c) uniform distribution, (b),(d) zipfian distribution.

7 CONCLUSION

In this paper, we proposed an efficient approach to evaluate dynamic group trip planning (DGTP) queries for the Euclidean space. Extensive set of experiments on real and synthetic datasets shows that our approach is on the average 99.74% faster and requires 94% less I/Os than the naive approach. We developed novel pruning techniques to refine the POI search space by exploiting elliptical properties, and dynamic programming techniques to compute the DGTP query answer that minimizes the aggregate travel distance for a dynamic group. In the future, we aim to extend our approach to handle the dynamic change of POI types such as visiting a new POI type in real time after the start of the group trip.

REFERENCES

[1] Elham Ahmadi and Mario A. Nascimento. 2015. A Mixed Breadth-Depth First Search Strategy for Sequenced Group Trip Planning Queries. In *MDM*. 24–33.
[2] Tanzima Hashem, Sukarna Barua, Mohammed Eunus Ali, Lars Kulik, and Egemen Tanin. 2015. Efficient Computation of Trips with Friends and Families. In *CIKM*. 931–940.
[3] Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, and Lars Kulik. 2013. Group Trip Planning Queries in Spatial Databases. In *SSTD*. 259–276.
[4] Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, Lars Kulik, and Egemen Tanin. 2016. Trip Planning Queries for Subgroups in Spatial Databases. In *ADC*. 110–122.
[5] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On Trip Planning Queries in Spatial Databases. In *SSTD*. 273–290.
[6] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. 2008. The optimal sequenced route query. *The VLDB Journal* 17, 4 (2008), 765–787.

[1]https://www.cs.utah.edu/ lifeifei/SpatialDataset.htm