Spatial Consensus Queries in a Collaborative Environment

MOHAMMED EUNUS ALI, Bangladesh University of Engineering and Technology, Bangladesh
EGEMEN TANIN, University of Melbourne, Australia
PETER SCHEUERMANN, Northwestern University, USA
SARANA NUTANONG, City University of Hong Kong, Hong Kong
LARS KULIK, University of Melbourne, Australia

We introduce a new type of query for a location based social network platform. Consider a scenario in which a group of users is trying to find a common meeting location, but attempting to include all group members is introducing a significant traveling cost to most of them. In this paper, we formulate a new query type called *the consensus query*, which can be used to help users explore these tradeoff options in order to find a solution upon which everyone can agree. Specifically, we study the problem of evaluating consensus queries in the context of nearest neighbor queries, where the group is interested in finding a meeting place that minimizes the travel distance for at least a specified number of group members. To help the group in selecting a suitable solution, the major challenge is to find optimal subgroups of all allowable subgroup sizes (i.e., greater or equal to the minimum specified subgroup size) that minimize the travel distances. We develop incremental algorithms to evaluate in one pass the optimal query subgroups of different sizes along with their corresponding nearest data points. These subsets which are evaluated by the location based service provider constitute the answer set that is returned to the group. The group then collaboratively selects the final answer from the candidate answer set. An extensive experimental study shows the efficiency and effectiveness of our proposed techniques.

Categories and Subject Descriptors: H.2 [**Database Management**]: Spatial Databases

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Consensus queries, location based services, group queries

1. INTRODUCTION

The development of location based social computing (e.g., [FourSquare 2011; Loopt 2011; iGroups 2011]) has introduced a new platform for accessing information *collaboratively* based on the current locations of participating users. Related social networking applications allow users to share their locations with others which enable a group of users to collaboratively search for an object of interest (e.g., a meeting place) that best suits the group. In such queries, a group of users needs to be actively involved in the query evaluation and the answer selection process. Depending on the circumstances, the group may opt for an answer that may not be optimal for all group members but suits most of them. Example applications for such queries include finding a suitable movie theater for a group of friends to go out on a Friday night or finding a shopping location for group of colleagues to buy a birthday present. Another example application for such a query is choosing a meeting location that is convenient for majority of the share holders in a corporation. In the multimedia domain, finding an image that is similar to a subset of given query images is another interesting application of our query.

Author's addresses: M. E. Ali is with the Department of CSE, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, E-mail: eunus@cse.buet.ac.bd; E. Tanin and L. Kulik are with Department of CIS, University of Melbourne, Australia; P. Scheuermann is with Department of EECS, Northwestern University, USA; S. Nutanong is with Department of CS, City University of Hong Kong, China.

DOI : http://dx.doi.org/10.1145/0000000.0000000

Let us consider a simple scenario where a group G of five users wants to select a location from a set O of all possible shopping locations to buy a birthday gift for a friend. Given that all five users need to be included in the gift selection process, one may consider this scenario as a group or aggregate nearest neighbor problem [Papadias et al. 2005], i.e., finding a location p in O which minimizes the maximum value of $\{|q-p| : q \in G\}$. As a result, the farthest distance that any user in this group of five needs to travel is minimized. In reality, however, one may wish to allow some users to be excluded if that substantially reduces the distance that most people have to travel, or one may wish to find out the maximum group size with the distance not exceeding a threshold.

In this paper, we address the above problem by (i) relaxing the constraint which dictates that all users in G must be included, and (ii) considering the number of users to be included as another optimization criterion (in addition to the distance). We also propose a novel query type called *consensus query* to enable users to incrementally browse possible solutions.

Figure 1 shows an example of our scenario - where the locations of the five group members $\{q_1, q_2, q_3, q_4, q_5\}$ are query points and the locations of seven possible meeting places $\{a, b, c, d, e, f, g\}$ are data points. In this example, if the locations of all group members are considered, the data point d is the query answer since d can be reached within the minimum possible travel time ($|d-q_3|$) by all group members. That is, in this case d minimizes the maximum travel time of all group members (assuming the travel time is proportional to the travel distance). Although the solution is optimized with respect to all members in the group, the fact that we try to accommodate all users in the group incurs significant traveling cost to most of the users. Specifically, in order to accommodate q_4 and q_5, the rest of the users (q_1, q_2, and q_3) need to travel long distances away from their nearest data points. By allowing certain users to be excluded, the group may opt for a meeting place that is optimal for a subgroup of *three* and is *acceptable* for the group in general (i.e., other members can skip or join later). Figure 1 shows an example where f is the more appropriate answer for the subgroup (q_1, q_2, q_3) since their minimum aggregate travel time $|f-q_3|$ is much less than $|d-q_3|$, the minimum aggregate travel time for the whole group. Similarly, the group may opt for the data point e, if the group can afford to delay the shopping a bit to allow the participation of four group members q_1, q_2, q_3, and q_4 with minimum aggregate travel time $|e-q_2|$.

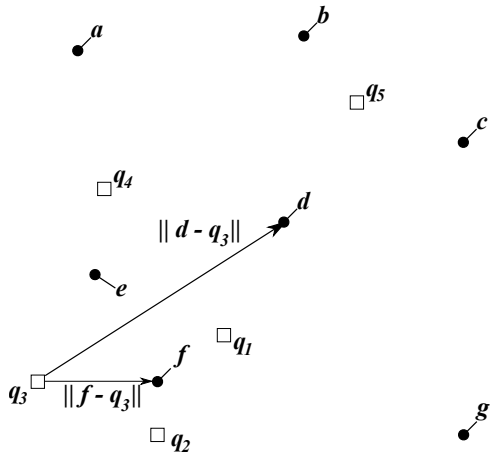

Fig. 1: A set of user locations $\{q_1,q_2,...,q_5\}$ and a set of data points $\{a,b,...,g\}$. d is the best answer that minimizes the maximum travel time when all 5 group members are taken into account, whereas f is the best answer when 3 out of 5 members are considered.

As we can see from our example, a group can opt for a solution that is optimum for at least a specified minimum $n' < n$ number of group members (i.e., three in our last example). In this case, the main task is to find the best subgroup among many subgroups which satisfy the minimum subgroup size requirement. However, the group may also wish to maximize the participation of members in the social activity. In that case, the group may want to explore other optimal solutions for subgroups that involve higher number of group members. Our motivation for expressing the subgroup size as a minimum requirement rather than a fixed value is based on the assumption that users prefer the number of participants to be high and the aggregate traveling cost to be low. Presenting the best (cheapest) subgroup for each subgroup size allows the users to view the results in terms of a trade off between the number of participants and traveling cost. In our current example shown in Figure 1, if we require at least three participants then the solution for any *subgroup* of three, four, or five users can be collectively selected as the *best* answer by the group.

In order to help the group decide on the participation of group members that may yield the best results for the group, the members first need to know a list of *candidate* solutions, i.e., the solutions with respect to different possible subgroup sizes. For this, the group first sends a query request to *the location based service provider (LSP)*. After receiving the candidate answer set from the LSP, the group collaboratively selects the final answer. In this paper, we call such a query, a *consensus query*. We study the consensus query in the larger context of the nearest neighbor (NN) query, which is a common query type in current location based applications.

It is important to notice here that in practice the LSP could select one of these candidate sets as the best answer for the group, but this is not a desirable solution from the standpoint of privacy as the group may not want to disclose the meeting location to the LSP. In our scenario, since the group asks for candidate answers sets for all possible subgroups, the LSP cannot identify the subgroup and the meeting location, which ensure the privacy of users. Moreover, the group may employ different policies to decide which of the candidate answers to choose, using different objective functions, as is the case in social networks [Gartrell et al. 2010]. In this paper, we focus on the query processing strategy at the LSP that finds a candidate answer set from a large number of data points, and then the group may choose any of the exiting objective functions [Gartrell et al. 2010; Jameson and Smyth 2007; Yu et al. 2009] to reach a consensus on the final answer from the candidate set.

The consensus query approach adopts the concept of exploratory querying, which is quite common in modern database systems (e.g., [Ferré and Hermann 2011; Shneiderman 1994; Kementsietsidis et al. 2008]). Exploratory querying allows users to find/navigate information without a priori knowledge of the data. A common practice in exploratory querying is to submit hundreds of simple queries that actually map to one single mental query submitted to the server for exploration. In our consensus query, we submit one query that essentially consists of multiple query instances (e.g., NNs for all subgroups) returning different result sets. The consensus query is preferred over the single query approach in many scenarios. For example, users may often be interested in soft/flexible constraints instead of hard constraints in the query, e.g., users specifying a traveling cost budget X may be ready to accept a solution with cost $X + \varepsilon$, where ε is small, if it maximizes the group size.

The main challenge for a consensus query evaluation by the LSP is to *identify* for each subgroup category (i.e, subgroups of same size) the subgroup that yields the optimum result. In order to support consensus queries, we define a new type of query in this paper, namely the BEST-SUBGROUPS-NN query. The answer to a BEST-SUBGROUPS-NN query consists of pairs of the form (o_m, sg_m) where o_m is the data object that minimizes the aggregate distance function for the subgroups of size m and sg_m is the corresponding subgroup. In addition, we impose the constraint that $n' <= m <= n$, with n' being the minimum number of users required in a subgroup.

A possible solution is to utilize the group NN querying technique of [Papadias et al. 2005]. Since their group NN query computes the aggregate distance using all of the query points in G, we have to combinatorially enumerate all possible subgroups that contain at least the specified number of members. We then need to find the NN for each of these subgroups using the group NN query technique of [Papadias et al. 2005]. This combinatorial approach incurs high query processing costs since each subgroup needs to be evaluated independently. In addition, this technique requires evalu-

ations of redundant subgroups that have no possibility of becoming a candidate for the final answer. These problems constitute a major bottleneck when the number of possible subgroups is large. Recently, Li et al. [Li et al. 2011] proposed a flexible aggregate nearest neighbor (FANN) algorithm that finds the nearest data object for a fixed subgroup size. To evaluate a consensus query using their approach would require running independent FANN queries for all subgroup sizes, which can be both computationally and I/O expensive.

In this paper, we propose an *incremental* solution that alleviates the above bottlenecks for processing a BEST-SUBGROUPS-NN query. In our proposed incremental approach as soon as we retrieve a data point, the optimal subgroups for all subgroup categories are updated based on the current knowledge of data points already retrieved. In order to answer a BEST-SUBGROUPS-NN query with reduced I/O costs, we propose two pruning rules that allow for early termination of the algorithm under tight and relaxed lower performance bounds.

In summary, our contributions are as follows:

— We formulate the problem of a consensus query in a collaborative environment.
— We develop an efficient algorithm to evaluate the BEST-SUBGROUPS-NN query for recommending optimum subgroups for a consensus query in the context of NNs.
— We conduct an extensive experimental study to show the effectiveness of our approach.
— We analyze the complexity of our proposed methods and competitive methods, and theoretically prove that our proposed method is superior to all competitive methods (see Appendix A.2).

The rest of the paper is organized as follows. Section 2 discusses preliminaries and the problem setup. Section 3 reviews the related work. In Section 4, we describe our algorithm for processing consensus queries. Section 5 reports our experimental results and Section 6 concludes the paper.

2. PROBLEM SETUP

Let O be a finite set of data point objects in a d-dimensional metric space and G be a set of n query points $\{q_1, q_2, \ldots, q_n\}$ representing a group of users. The group issues a query to the LSP for the nearest data objects that minimize the aggregate function f (SUM, MAX, or MIN) for each subgroup size m, where $n' \leq m \leq n$ and n' is the required minimum number of users to form a subgroup. First we define the group nearest neighbor (GNN) query and BEST-SUBGROUP(m)-NN query.

Definition 2.1. (GNN). Given a set O of objects, a set G of query points $\{q_1, q_2, \ldots, q_n\}$, and an aggregate function f, the GNN query finds an object o such that for any $o' \in O - \{o\}$, $f(\{|q-o| : \forall q \in G\}) \leq f(\{|q-o'| : \forall q \in G\})$.

Since n is the total number of users in the group, there are $\binom{n}{m}$ possible subgroups of size m. We call these subgroups the *category m subgroups* in this paper. We define the BEST-SUBGROUP(m)-NN query as a query which finds the subgroup from the possible $\binom{n}{m}$ subgroups along with the data object that results in a minimum value for the aggregate function on the subgroup's query points. A formal definition of this query is given as follows.

Definition 2.2. BEST-SUBGROUP(m)-NN. Given a set O of objects, a set G of query points $\{q_1, q_2, \ldots, q_n\}$, an aggregate function f, a subgroup size m, and the set SG_m of all possible subgroups of size m, the BEST-SUBGROUP(m)-NN query finds a subgroup $sg_m \in SG_m$ and an object $o \in O$ such that for any $o' \in O - \{o\}$, $f(\{|q-o| : q \in sg_m\}) \leq f(\{|q-o'| : q \in sg_m\})$ and for any subgroup $sg'_m \in SG_m - \{sg_m\}$ and for any $o' \in O$, $f(\{|q-o| : q \in sg_m\}) \leq f(\{|q-o'| : q \in sg'_m\})$.

Based on Definitions 2.1 and 2.2, we can formally define the BEST-SUBGROUPS-NN query.

Definition 2.3. BEST-SUBGROUPS-NN. Given a set O of objects, a set G of query points $\{q_1, q_2, \ldots, q_n\}$, an aggregate function f, and the minimum subgroup size n', the BEST-SUBGROUPS-NN query finds a set A of (subgroup, data object) pairs such that (i) A contains $(n-n'+1)$ pairs; (ii) each pair (sg_m, o_m) is the result of the BEST-SUBGROUP(m)-NN query, where $n' \leq m \leq n$.

The BEST-SUBGROUPS-NN query finds the best subgroups for each subgroup size m, where $n' \leq m \leq n$. We generalize Definition 2.3 for the k-BEST-SUBGROUPS-NN query that finds the top k subgroups from each subgroup category. For this, we first define the k-BEST-SUBGROUP(m)-NN query that finds the k best subgroups of the subgroup category m and the associated NNs.

Definition 2.4. k-BEST-SUBGROUP(m)-NN. Given a set O of objects and a set G of query points $\{q_1, q_2, \ldots, q_n\}$, an aggregate function f, the subgroup size m, and the set S of all possible subgroups of size m. The k-BEST-SUBGROUP(m)-NN query finds a list L_m such that (i) L_m contains k pairs, $\{\langle sg_m^1, o^1\rangle, \langle sg_m^2, o^2\rangle, ..., \langle sg_m^k, o^k\rangle\}$, where $sg_m^i \in SG'_m$ ($\subseteq SG_m$) and $o^i \in O$ for $1 \leq i \leq k$; (ii) for any adjacent pairs $\langle sg_m^i, o^i\rangle$ and $\langle sg_m^{i+1}, o^{i+1}\rangle$ in the list L_m $f(\{|q - o^i| : q \in sg_m^i\}) \leq f(\{|q - o^{i+1}| : q \in sg_m^{i+1}\})$; (iii) for any pair $\langle sg_m^i, o^i\rangle \in L_m$ and any other pairs $\langle sg_m^j, o^j\rangle$ where $sg_m^j \in SG_m - SG'_m$ and $o^j \in O$, $f(\{|q - o^i| : q \in sg_m^i\}) \leq f(\{|q - o^j| : q \in sg_m^j\})$.

Finally, we can formally define k-BEST-SUBGROUPS-NN query as follows.

Definition 2.5. k-BEST-SUBGROUPS-NN. Given a set O of objects and a set G of query points $\{q_1, q_2, \ldots, q_n\}$, an aggregate function f, and the minimum subgroup size n', the k-BEST-SUBGROUPS-NN query finds a set A of lists $\{L_{n'}, L_{n'+1}, ..., L_n\}$ such that, for $n' \leq m \leq n$, (i) each L_m contains k pairs; (ii) each L_m is the result of the k-BEST-SUBGROUP(m)-NN query.

We consider the three most common aggregate functions SUM (or AVG), MAX, and MIN that minimize the total distance, the maximum distance, and the minimum distance from the users of a group to a data point, respectively.

3. RELATED WORK

Location based social networking applications [FourSquare 2011; Loopt 2011; Gowalla 2011] are upcoming platforms for interacting and socializing with friends. In order to capitalize on the potential of new social platforms, recent research has focused on developing efficient techniques that recommends different search results by combining the location and social context of users [Zheng et al. 2010; Chow et al. 2010; Sakaki et al. 2010; Si et al. 2010]. The consensus query that we define in this paper, utilizes the collaborative nature of location based social networking applications.

The problem of searching for a nearby data object from a database based on the locations of a single user or a group of users has received considerable attention from the database community (e.g., [Papadias et al. 2005; Hjaltason and Samet 1995; Roussopoulos et al. 1995; Yiu et al. 2005; Masud et al. 2013]). When a user searches for the nearest data object with respect to her own location (a query point) only, the query is known as the nearest neighbor (NN) query (e.g., [Hjaltason and Samet 1995; Roussopoulos et al. 1995]). The generalization of the NN query is known as the kNN query, where the user is interested in the k nearest data objects from her current location.

Existing techniques for processing kNN queries assume that the data objects are indexed, e.g., using an R-tree [Guttman 1984]. In order to find the k NNs of a query point, the tree can be traversed in a depth-first (DF) [Roussopoulos et al. 1995] or a best-first (BF) [Hjaltason and Samet 1995] manner. In the BF technique, the search starts from the root of the R-tree. Initially, all child nodes of the root are stored in a priority queue. The entries in the priority queue are ordered based on the minimum distance between the query point and the minimum bounding rectangles (MBRs) of R-tree nodes or data objects. In the next step, the algorithm removes the top element from the queue, which is the node representing the MBR or data object with the minimum distance from the query point. If the removed element is a node, the algorithm again inserts the child nodes or data objects of the removed node into the priority queue. On the other hand if the dequeued item is a data object, then the corresponding object is reported as the next nearest neighbor. The above process continues until k data objects, i.e., the k NNs, are dequeued from the queue.

Both DF and BF search techniques are widely used as the bases to process variants of NN queries [Shan et al. 2003; Seidl and Kriegel 1998]. NN queries have also been studied in road

networks [Papadias et al. 2003; Jensen et al. 2003; Kolahdouzan and Shahabi 2004], where the distance between two points is computed as the shortest path connecting those points.

When a group of users is involved in a query that searches for the nearest data object with respect to the locations (a set of query points) of all users in the groups the query is known as the group nearest neighbor (GNN) query (e.g., [Papadias et al. 2005; Yiu et al. 2005; Papadias et al. 2004]). A GNN query minimizes an aggregate function such as SUM, MAX, or MIN for the group. The generalization of the GNN query is known as the *k*GNN query, in which the group wants to find k GNNs for the group. Three different techniques to evaluates GNN queries for the aggregate function SUM were introduced in [Papadias et al. 2004]: multiple query method (MQM), single point method (SPM), and minimum bounding method (MBM). Later in [Papadias et al. 2005], these techniques were extended for the aggregate functions MAX and MIN.

MQM incrementally searches for the nearest data point to each query point in the set and computes the aggregate distance from all query points for each retrieved data point. The k GNNs are determined by combining the GNNs of previously retrieved data points. The search ends when it is ensured that the aggregate distance of any non-retrieved data point in the database is greater than the current *k*th minimum aggregate distance. The disadvantage of MQM is that it traverses the *R*-tree multiple times and may access the same node or a data point more than once. To avoid these limitations, SPM and MBM find the k GNNs in a single traversal of the *R*-tree. SPM starts the search from the centroid of the query set and incrementally accesses nearest data points from the centroid. For each retrieved data point, SPM computes the aggregate distance to the data point from all query points. The process continues until the actual k GNNs are determined. While SPM visits the *R*-tree nodes and data points in order of their minimum distances from the centroid, MBM visits *R*-tree nodes in order of the minimum aggregate distance from all query points. Therefore, MBM only visits the nodes that may contain candidate data points.

A generalization of the GNN query, called Group Nearest Group (GNG) query, was proposed by Deng et al. [Deng et al. 2012]. A GNG query finds a subset s of data points, instead of a single data point in GNN, from the database such that the aggregate distance from the group to s is minimum. Another variants of the GNN query, namely socio-spatial group query (SSGQ) has been proposed in [Yang et al. 2012]. Given a set of n users and their locations in a location based social network graph, a meeting point p, and a social constraint k, SSGQ finds a group of m users such that no selected user has more than k unfamiliar users in the selected group and the average distance for each selected user to p is minimized. Recently, Hashem et al. [Hashem et al. 2013] propose a method for a group users to plan a trip with a minimum aggregate trip distance, where the trip starts from the source locations of the group members, goes via different categories of data points such as a restaurant, shopping center and movie theater, and ends at the destination locations of the group members.

While the works of [Papadias et al. 2005; Papadias et al. 2004] focus on an Euclidian space, [Yiu et al. 2005] study the problem of GNN queries in a road network. This algorithm is based on an incremental expansion of the Euclidean NNs followed by computation of their aggregate network distance until the results are obtained.

A work whose motivation is similar to ours is the one proposed in [Li et al. 2011], which introduced the concept of a flexible aggregate nearest neighbor query (FANN). Given a set of users G, a FANN query returns as result a tuple (d, sg_m), where d is an object in the database that minimizes the distance to a fraction of the users, i.e., a subgroup of fixed size m denoted as sg_m. The authors of [Li et al. 2011] propose two exact methods to search for the desired solution: one based on the traditional branch-and-bound method using an R-tree, and the second one based on a merge algorithm for sorted lists of objects in the database ranked by their proximity to a given query point (user). In addition, the paper also discussed a number of approximation algorithms for various aggregate functions. We observe here that if we were to use the FANN method to solve a consensus query (i.e., BEST-SUBGROUP-NN) we will need to run independent FANN queries for all subgroups of size m where, $n' <= m <= n = |G|$.

In addition to NN queries, the body of work on query processing for road-networks includes also a different paradigm, namely optimal location queries. These types of queries [Xia et al. 2004] are related to the facility location problem, namely given a set of clients (users), a set of facilities and a set of candidate locations they identify a candidate location where a new facility can be added so as to minimize a given cost metric. However, the techniques proposed to solve optimal location queries are not compatible with the ones needed for NN queries.

With the proliferation of Geo-Spatial Networks (GSN) it is more common to encounter geo-tagged objects, i.e., objects associated with some textual description in addition to geographical coordinates. In this context spatial keyword queries have been studied extensively in the past few years [Chen et al. 2013]. A kNN spatial keyword query retrieves the k nearest objects to a given query point whose set of keywords cover the ones given in the query [Felipe et al. 2008]. Alternatively, top-kNN spatial keyword queries [Rocha-Junior and Nørvåg 2012] find the top-k ranked objects where the ranking score is a combination of geographical proximity and textual relevance. On the other hand, spatial group keyword queries find a group of objects that collectively satisfy the query keywords and minimize the total distance to the query point [Cao et al. 2011]. We note that for both queries a single user is involved in posing the queries. Zhang et al. [Zhang et al. 2012] have recently introduced the concept of a top-k collaborative spatial keyword query where the set of keywords comes from a number of originators (users). However, as with all other top-kNN query approaches, the result consists of only one answer set, in contrast to our approach where we present the users with a number of candidate answers for all possible subgroups of users.

4. OUR APPROACH

In order to answer a consensus query, we need to evaluate a k-BEST-SUBGROUPS-NN query that finds the best k subgroups in each subgroup category (i.e., subgroups of the same size) based on the aggregate distances of these subgroups to the data points. A naive approach to evaluate a k-BEST-SUBGROUPS-NN query would require enumerating all subgroups, and then finding the kNNs for each subgroup using an existing group query processing technique such as those used for GNN queries. After finding the kNNs for all subgroups, a set of subgroups, one from each subgroup category that minimizes the aggregate function, is selected as the answer to the k-BEST-SUBGROUPS-NN query.

Without loss of generality, consider our initial example where a group of 5 users issue a k-BEST-SUBGROUPS query with minimum allowed subgroup size $n' = 3$ and $k = 1$ for the aggregate function MAX. Figure 1 shows the locations of the five users $\{q_1, q_2, ..., q_5\}$ and the data objects $\{a, b, ..., g\}$. To find the answer to the BEST-SUBGROUPS-NN query, the naive approach would require us to independently evaluate each of the 16 subgroups (10 subgroups of size 3, 5 subgroups of size 4, and 1 subgroup of size 5), and select the best answer.

The above approach incurs high query processing overhead since each subgroup needs to be evaluated independently. Moreover, this straightforward technique evaluates redundant subgroups that have no possibility of becoming a candidate for the final answer. These problems become a major bottleneck for a large number of possible subgroups.

The flexible aggregate nearest neighbor query (FANN), proposed by Li et al. [Li et al. 2011], finds a subgroup and an object that minimize the aggregate distance for a fixed subgroup size. Instead, we propose efficient algorithms for processing a consensus query where the subgroup size is given as a minimum requirement rather than a fixed value. For example, given a minimum subgroup size, one may wish to find out the maximum group size with the distance not exceeding d_{max}. A direct application of FANN to the query in this example requires execution of multiple instances of FANN for all possible subgroup sizes. Furthermore, since the proposed FANN algorithms rely on the assumption that the group size is fixed we cannot directly extend their algorithms to support our consensus query. We propose efficient algorithms, which incrementally explore different subgroup sizes in a single query. As shown by our experimental results in Section 5, our proposed incremental algorithms scale much better than FANN as the number of considered subgroup size increases.

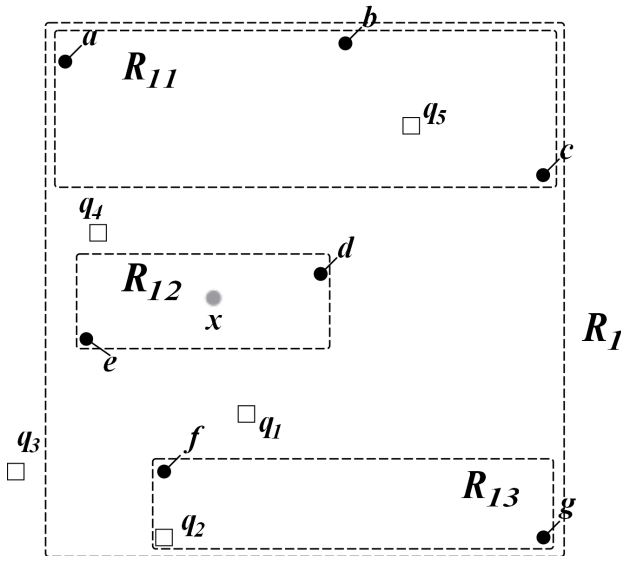

Fig. 2: An R-tree node R_1 that encloses data objects $\{a, b, ..., g\}$. R_{11}, R_{12}, and R_{13} are three child nodes of R_1. Five query (users) locations $\{q_1, q_2, ..., q_5\}$ and their centroid x are also shown.

To overcome the above limitations, we propose a data-centric approach that incrementally accesses the data objects and identifies the best subgroups in a single pass. The key idea of our data-centric approach is to form the best subgroups with respect to existing data points by advancing in a radial fashion from the centroid of the query points, which eliminates the need to enumerate all possible subgroups.

We first discuss our approach using the running example and then present our algorithm in detail.

4.1. Incremental Expansion

In our approach, since we want to find the result in a single pass from the database, we need to choose an appropriate point x to start the search so that the data retrieval cost is kept low. Though any arbitrary starting point will ensure the correctness of the algorithm, we consider the *centroid* that corresponds to the center of the smallest disk containing all the query points as the starting point of the search. The rationale behind the above centroid instead of the geometric centroid is as follows. Our search technique expands the search space in a radial fashion and it needs to discover at least the data points that fall inside the search space covering all the query points to ensure the correct answer Since our chosen centroid requires the minimum possible area expansion to include all query points, it results in fewer I/Os compared to geometric centroid-based data retrieval. The centroid can be computed by an existing solution for the minimum enclosing circle problem (e.g., [Welzl 1991]).

An alternative starting point of the search could be a biased point which constitutes the centroid of a subgroup of query points. However, in real-world scenarios, there could be many such clusters (i.e., subgroups) distributed in the dataspace, and thereby, starting from the centroid of a single cluster may potentially lead to worse search performance while finding optimal answers for all subgroups.

Our proposed solution has three key steps: retrieve data points incrementally in a radial fashion, calculate the best subgroups with respect to the retrieved data points, and use a pruning test based on computational geometry in order to terminate the algorithm as soon as the correct answer is guaranteed.

For each retrieved data object, we update the list of pairs (subgroup, data object) that minimize the aggregate function with respect to the data points processed so far. The process continues until the best subgroups with respect to the entire database are discovered. A major contribution of our approach is the pruning rules based on computational geometry arguments that guarantee early termination of the algorithm and give us substantial savings in terms of data access and computational costs.

Let us consider again our running example from Figure 1 with the data objects $\{a,b,...,g\}$ and the query objects $\{q_1,q_2,...,q_5\}$. We want to find the answer to the BEST-SUBGROUPS-NN query for the aggregate function MAX, with $k=1$ and $m=3$. Since our data objects are indexed using an R-tree we illustrate in Figure 2 an R-tree node R_1 that encloses data objects $\{a,b,...,g\}$. R_1 has three child nodes R_{11}, R_{12} and R_{13}. Our approach incrementally accesses the data objects in increasing order of their distances from a point x. We use the best-first (BF) technique [Hjaltason and Samet 1995] to incrementally retrieve the nearest data objects with respect to x from the R-tree.

In the BF technique, the search starts from the root node of the R-tree, and a priority queue is maintained where retrieved R-tree nodes (or MBRs) and data objects are kept in increasing order of their distances from x. Here, the distance from x to a data point is measured as the Euclidean distance between these two points, and the distance from x to an MBR is computed as the minimum distance between x and the MBR. Let us assume that R_1 is the top element in the priority queue. In the next step R_{11}, $R12$, and R_{13} are retrieved from the index and pushed into the queue yielding $[R_{12},R_{13},R_{11}]$, since R_{12} is the MBR whose distance from x is minimal among the three R-tree nodes. Next, R_{12} is removed from the top of the heap and objects d and e are inserted into the priority queue yielding $[d,e,R_{13},R_{11}]$. Next objects d and e are removed from the priority queue and since they have no descendants no new elements are added to the queue resulting in $[R_{13},R_{11}]$. At this stage the MBR corresponding to R_{13} is deleted and the objects that it encloses , namely f and g are inserted into the queue resulting in $[f,R_{11},g]$. Following the deletion of f and R_{11} from the queue we need to insert the objects enclosed in R_{11} and object a ends up at the top of the heap. This process continues in similar fashion for the remaining data object. As soon as a data object is removed from the queue, we determine the best subgroups with respect to the data objects as outlined in subsequent paragraphs.

Table I shows the steps involved in the best subgroups computation, where the last cell (last row and last column) contains the optimal subgroups, $\{q_2,q_1,q_3\}$, $\{q_4,q_3,q_1,q_2\}$, $\{q_1,q_5,q_4,q_2,q_3\}$, and the corresponding data objects f, e, d of the subgroups of size 3, 4, and 5, respectively. The details of each step are given as follows.

Step 1. As we can see from Figure 3, d is accessed first, since it is the nearest data object to x. After accessing d, we compute the distances of all query points, $q_1,q_2,...,q_5$, to d and store all (query, distance) pairs in an ordered list Q (e.g., a priority queue in order of their minimum distance to d) as shown in the first row of the third column of Table I.

Then, the best subgroups with respect to the current retrieved data objects, termed as current best subgroups (CBSG) are computed and stored in a list dG. To find the CBSG of size 3 with respect to d, we retrieve the top 3 elements from Q and estimate the min-max distance as 42, i.e., $42=\max(29,32,42)$ which is achieved for the CBSG, $\{q_1,q_5,q_4\}$, the one having a minimum aggregate distance to x. The result $(\{q_1,q_5,q_4\},42,d)$ is stored as the *CBSG* of size 3 (shown in the first entry in column dG of row 1 in Table I). Similarly, from the top 4 and 5 elements from Q, we compute the *CBSG*s, $(\{q_1,q_5,q_4,q_2\},55,d)$ and $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$ as the CBSGs of sizes 4 and 5, respectively, and store in the list dG.

As the last action in this step, we compute the best subgroups with respect to the *already retrieved objects*, and store them in the list dG' as shown in the last column in Table I. In this case, since only object d is retrieved, the list dG' is simply updated with the list dG.

Step 2. Data object e is retrieved next as it is the second nearest to x. Note that we omit there the intermediate MBRs retrieved from the index as discussed earlier. Then, we update Q and dG for object e by using a similar procedure to the one described in step 1. The third and fourth columns of the second row in Table I show the computed values for Q and dG, respectively. After that, each

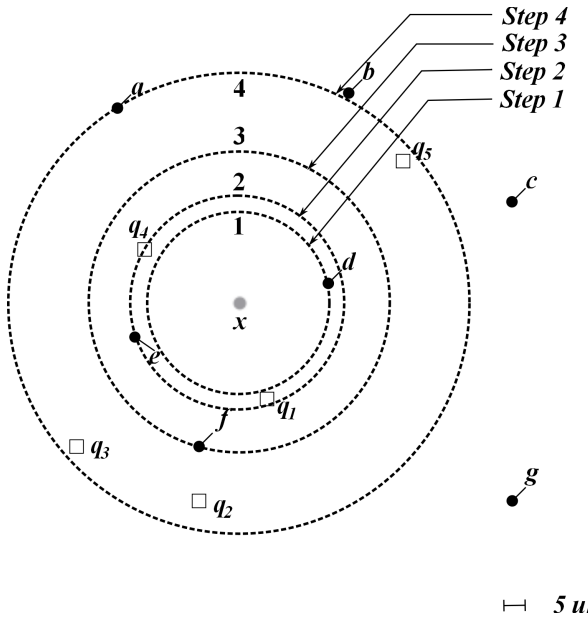

Fig. 3: The centroid x of the five query points $\{q_1, q_2, q_3, q_4, q_5\}$, the steps of accessing the data points $d, e, f, a, ...$ from x (the distance scale is shown at the lower right corner).

element of dG (i.e., dG of step 2) is compared with the corresponding element of the previously computed dG' (i.e., dG' of step 1), and the current dG' is updated if the aggregate distance of a dG subgroup is smaller then that of the subgroup of the same size in the previous dG'. In this case, the first two elements $(\{q_1, q_5, q_4\}, 42, d)$ and $(\{q_1, q_5, q_4, q_2\}, 56, d)$, i.e., the previously computed best subgroups of size 3 and 4 in dG' (of step 1), are replaced by the elements of dG computed in step 2, $(\{q_4, q_3, q_1\}, 33, e)$ and $(\{q_4, q_3, q_1, q_2\}, 39, e)$, respectively. This is because the subgroup $\{q_4, q_3, q_1\}$ results in aggregate distance 33 for object e, which is smaller than the previously computed aggregate distance 42 from the subgroup $\{q_1, q_5, q_4\}$ to d, and the aggregate distance 39 from the subgroup $\{q_4, q_3, q_1, q_2\}$ to d is smaller than the previously computed aggregate distance 55 from the subgroup $\{q_1, q_5, q_4, q_2\}$ to d.

Steps 3 and 4. The above process continues until all date objects necessary to compute the optimal subgroups for all allowable group sizes have been retrieved.

Although our search process is based upon accessing the data points incrementally, this process can be provided a boost by observing that our search space needs to cover at least all the data points inside the smallest disk containing all the query points. In Figure 3 this is the disk containing the data points $\{d, e, f, a\}$. We can implement a batch step which unravels a number of leaf nodes of the R-tree and stores the data points contained in this disk into a temporary buffer TB. After expanding R_{11} data points d and e are added to TB instead of the of the priority queue. Following the expansion of R13 the buffer TB will contain $\{d, e, f\}$ and the priority queue holds $\{R_{11}, g\}$. Note that TB does not contain data point g since g is outside the minimum spanning disk as shown in Figure 3. Similarly, after R_{11} is expanded TB consists of $\{d, e, f, a\}$ and the priority queue holds $\{b, c, g\}$. Finally we cannot add any more data points to TB since the first candidate in the priority queue, data point b, falls outside the minimum spanning disk. At this point we can do a batch search on disk for all the data points contained in TB and after this boost up we continue accessing the data points incrementally using only the priority queue as illustrated earlier in this section. We conjecture

that this batch process can be beneficial if the query points are far apart and the minimum spanning disk contains a large number of data objects.

Step	Data Object	Q	dG	dG'
1	d	$(q_1,29)$ $(q_5,32)$ $(q_4,42)$ $(q_2,56)$ $(q_3,66)$	$(\{q_1,q_5,q_4\},42,d)$ $(\{q_1,q_5,q_4,q_2\},56,d)$ $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$	$(\{q_1,q_5,q_4\},42,d)$ $(\{q_1,q_5,q_4,q_2\},56,d)$ $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$
2	e	$(q_4,21)$ $(q_3,28)$ $(q_1,33)$ $(q_2,39)$ $(q_3,71)$	$(\{q_4,q_3,q_1\},33,e)$ $(\{q_4,q_3,q_1,q_2\},39,e)$ $(\{q_4,q_3,q_1,q_2,q_5\},71,e)$	$(\{q_4,q_3,q_1\},33,e)$ $(\{q_4,q_3,q_1,q_2\},39,e)$ $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$
3	f	$(q_2,12)$ $(q_1,19)$ $(q_3,27)$ $(q_4,44)$ $(q_5,78)$	$(\{q_2,q_1,q_3\},27,f)$ $(\{q_2,q_1,q_3,q_4\},44,f)$ $(\{q_2,q_1,q_3,q_4,q_5\},78,f)$	$(\{q_2,q_1,q_3\},27,f)$ $(\{q_4,q_3,q_1,q_2\},39,e)$ $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$
4	a	$(q_4,32)$ $(q_5,64)$ $(q_1,72)$ $(q_3,75)$ $(q_2,88)$	$(\{q_2,q_1,q_3\},72,a)$ $(\{q_2,q_1,q_3,q_4\},75,a)$ $(\{q_2,q_1,q_3,q_4,q_5\},88,a)$	$(\{q_2,q_1,q_3\},27,f)$ $(\{q_4,q_3,q_1,q_2\},39,e)$ $(\{q_1,q_5,q_4,q_2,q_3\},66,d)$

Table I: The steps of finding the *Best-Subgroups-NN*s

4.2. Stopping Criteria and the Distance Metrics

The main challenge of the above technique is how to determine the condition when the search can safely terminate. Intuitively, the search can terminate when we know that objects outside the scope of our current incremental radial expansion will not affect the optimal subgroups of any size. That is, we need to explore the database as long as there is a possibility that an unexplored object can be a candidate of the answer set. In this subsection, we derive two types of termination conditions. The first type minimizes the data retrieval cost and requires complex calculations. The second type, on the other hand, requires a simpler calculation but incurs a greater data object retrieval cost than the first.

We now discuss the distance metrics which form the basis of our two pruning methods that allow us to terminate the data search early. We define two distance metrics: *best known aggregate distance (BKAD)* and *best unknown aggregate distance (BUAD)* based on the retrieved data objects and the known region. The *known region* is a region centered at the query centroid x for which all objects inside this region are already retrieved from the database. Based on the BKAD and BUAD, we give lemmas to define the termination conditions of our algorithm.

Let G be the set $\{q_1,q_2,\ldots,q_n\}$ of n query points representing the locations of all the users in the group, and SG_m be the set of $\binom{n}{m}$ unique subgroups of category m, where each subgroup $sg_m^i \in SG_m$ comprises m query points from G, with $1 \leq i \leq \binom{n}{m}$. We use the concept of *known region* to derive the two pruning rules of our algorithm. We term a circular region $C(x,r)$ centered at a point x with a radius r, as the known region if the locations of all the data points inside this region are known i.e., all the objects inside this region were retrieved already from the database. Since we incrementally retrieve data points from an R-tree in increasing order of their distances from point x, the value of r is initially set to the Euclidian distance between x and the *nearest* data point or MBR of an R-tree

node. In subsequent steps, the value of r is updated with the Euclidian distance between x and the *last* retrieved data point or the MBR of an R-tree node. In our running example in Figure 3, with the last retrieved data point b, $C(x,r)$ is updated with the distance $|x-b|$.

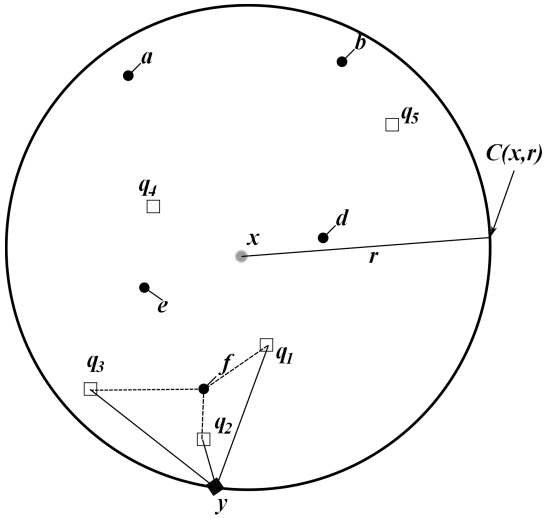

Fig. 4: The known region $C(x,r)$, the BKAD $|q_3-f|$ and the BUAD $|q_1-y|$ of a subgroup $\{q_1,q_2,q_3\}$.

Based on the concept of circular known region, we can now define the best known aggregate distance (BKAD), $BKAD_m$, for the subgroup category m. We first define the BKAD for a single group sg_m^i of size m as $BKAD_m^i$, where $1 \leq i \leq \binom{n}{m}$.

Definition 4.1. (BKAD FOR A SUBGROUP). Given a set $sg_m^i = \{q_1,q_2,\ldots,q_m\}$ of m query points and a set O' of data objects inside a circular region $C(x,r)$ centered at a point x with a radius r, and an aggregate function f. The best known aggregate distance (BKAD) for the subgroup sg_m^i, $BKAD_m^i$, is the minimum aggregate distance of sg_m^i to a data object $o \in O'$, such that $f(sg_m^i,o) < f(sg_m^i,o')$ for any $o' \in O'-\{o\}$.

As shown in the above definition, finding the BKAD for a subgroup requires us to find a data point that minimizes the aggregate distance for the subgroup. Figure 4 shows an example, where f minimizes the aggregate distance for the subgroup $\{q_1,q_2,q_3\}$ for the aggregate function MAX, and $|q_3-f|$ is the BKAD for this subgroup.

Furthermore, the $BKAD_m$ can be defined as $\min_i BKAD_m^i$.

Let o'' be an object outside $C(x,r)$. It is possible that the aggregate distance to o'' from a subgroup $sg_m^i \in SG_m$ is smaller than the $BKAD_m$. Since the locations of objects (if any) outside $C(x,r)$ are unknown, we need to find the best *unknown* aggregate distance (BUAD), $BUAD_m$, for the subgroup category m to any object outside $C(x,r)$. In other words, we need to find a point y residing on the boundary point set ß of $C(x,r)$ that results in the best possible aggregate distance. We can formally define the BUAD for a group sg_m^i of size m as $BUAD_m^i$.

Definition 4.2. (BUAD FOR A SUBGROUP). Given a subgroup $sg_m^i = \{q_1,q_2,\ldots,q_m\}$ of m query points inside a circular region $C(x,r)$ centered at a point x with a radius r, and an aggregate function f. The $BUAD_m^i$, is the minimum aggregate distance from the subgroup sg_m^i to a point y on the boundary set ß of $C(x,r)$, such that

(i) $y \in \{v : |x-y| = r\}$;

(ii) y satisfies: $f(\{|q_j - y| : \forall q_j \in sg_m^i\}) \leq f(\{|q_j - z| : \forall q_j \in sg_m^i\})$, where $y, z \in$ ß.

As shown in the above definition, finding the BUAD for a subgroup requires us to identify a point $y \in$ ß that minimizes the aggregate distance for the subgroup. Figure 4 shows an example, where y minimizes the aggregate distance for the subgroup $\{q_1, q_2, q_3\}$ for the aggregate function MAX. Thus $|q_1 - y|$ is the BUAD for the subgroup $\{q_1, q_2, q_3\}$. For the subgroup category m, Then the $BUAD_m$, can be computed by taking the minimum of all $BUAD_m^i$s for subgroups $sg_m^i \in SG_m$, i.e.,$\min_i BUAD_m^i$.

From the above formulations of BKAD and BUAD, we can define the termination condition of our algorithm.

Definition 4.3. (TERMINATION CONDITION).
(i) The search for the best subgroup in category m is terminated when $BKAD_m \leq BUAD_m$.
(ii) The search for a BEST-SUBGROUPS-NN query is terminated when the above condition holds true for every subgroup category i, for all $n' \leq i \leq n$.

.

To determine the termination condition for a k-BEST-SUBGROUPS-NN query, the BKAD of a subgroup category is computed as the kth minimum aggregate distance of that subgroup category. The details of BUAD computation are given in Appendix A.1.

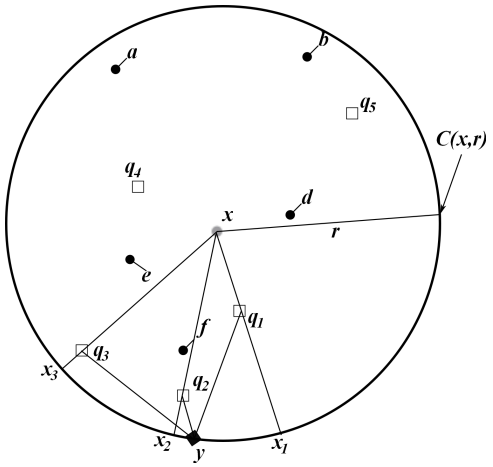

Fig. 5: The known region $C(x,r)$, the BUAD $|q_1 - y|$ and a relaxed lower bound $|q_2 - x_2|$ for a subgroup $\{q_1, q_2, q_3\}$. x_is are obtained by taking the intersection of the lines (x, q_i) with the boundary of the known region.

4.2.1. A relaxed lower bound (RBUAD). Finding the exact value for the BUAD of a subgroup category can be computationally expensive, since it requires searching for a location y from all possible points in the boundary point set ß $\equiv \{v : |o - y| = r\}$ and the process needs to be repeated for all possible subgroups of a category. Note that the BUAD is the tightest lower bound of an unknown aggregate distance for a set of query points. We will show how to find a relaxed *lower bound* for a BUAD, with reduced computational overheads, which is a more practical approach to use as the termination condition of our algorithm.

We define now a relaxed lower bound of the aggregate distance of any subgroup i of size m, denoted as $RBUAD_m^i$ as follows.

LEMMA 4.4. *Let $sg_m = \{q_1, q_2, \ldots, q_m\}$ be a set of m query points inside a circular region $C(x,r)$ centered at a point x with a radius r, an aggregate function f, and $qmin_i$ the minimum distance from q_i to the boundary point set ß of $C(x,r)$. Then $RBUAD^i_m = f(qmin_1, qmin_2, \ldots, qmin_m)$.*

PROOF. Let $y \in$ ß be a point that *minimizes* the aggregate distance for the subgroup sg_m. In this case, the BUAD can be expressed as $f(|q_1 - y|, |q_2 - y|, ..., |q_m - y|)$.

Now, let $x_i \in$ ß be a point that minimizes the distance from q_i to any boundary point set of $C(x,r)$. That is, $qmin_i$ is equal to $|q_i - x_i|$.nAccording to the triangle inequality, for any query point $q_i \in sg_m$, the following inequality holds.

$$|x - q_i| + |q_i - y| \geq |x - y|.$$

Since, $|x-y| = |x-q_i| + |q_i - x_i| = r$, we can replace $|x-y|$ with $|x-q_i| + |q_i - x_i|$ in the above equation and get the following equation:

$$\begin{aligned} &|x - q_i| + |q_i - y| \geq |x - q_i| + |q_i - x_i| \\ &|q_i - y| \geq |q_i - x_i|. \end{aligned}$$

If we consider m query points and apply our aggregate function f, we can have the following equation, $f(|q_1 - y|, |q_2 - y|, \ldots, |q_m - y|) \geq f(|q_1 - x_1|, |q_2 - x_2|, \ldots, |q_m - x_m|)$. That is, $f(|q_1 - y|, |q_2 - y|, \ldots, |q_m - y|) \geq f(qmin_1, qmin_2, \ldots, qmin_m)$.

Hence, $RBUAD^i_m = f(qmin_1, qmin_2, \ldots, qmin_m)$ gives a lower bound of the BUAD for the subgroup sg_m. □

Figure 5 shows the lower bound, $|q_1 - x_1|$, of the BUAD for three query points q_1, q_2, and q_3 for the aggregate function MAX. Based on the lower bound of a subgroup of size m as shown in Lemma 4.4, a lower bound for the BUAD of a subgroup category m, $RBUAD_m$ can be formally defined using the following lemma.

LEMMA 4.5. *Let $G = \{q_1, q_2, \ldots, q_n\}$ be a set of n query points inside a circular region $C(x,r)$ centered at a point x with a radius r, an aggregate function f, and $qmin_i$ the minimum distance from q_i to the boundary of $C(x,r)$. If min_j is the jth minimum from the set $\{qmin_1, qmin_2, \ldots, qmin_n\}$ of values, then $RBUAD_m = f(min_1, min_2, \ldots, min_m)$ gives a lower bound of the BUAD for all possible subgroups of size m.*

PROOF. The BUAD for a subgroup category can be computed by taking the minimum of all $RBUAD^i_m$ s of that category, i.e., $\min_{i=1}^{\binom{n}{m}}(RBUAD^i_m)$. Let us assume that the BUAD of subgroup sg^k_m, $RBUAD^k_m$ is the minimum of all BUADs, and the subgroup comprises the first m query points, $\{q_1, q_2, \ldots, q_m\}$. Thus, according to Lemma 4.4, we have $RBUAD^k_m \geq f(qmin_1, qmin_2, \ldots, qmin_m)$.

Since min_j is the jth minimum from the set $\{qmin_1, qmin_2, \ldots, qmin_n\}$ and the set $\{min_1, min_2, \ldots, min_m\}$ is comprised of the minimum m values from the set $\{qmin_1, qmin_2, \ldots, qmin_n\}$, we have $f(qmin_1, qmin_2, \ldots, qmin_m) \geq f(min_1, min_2, \ldots, min_m)$. Hence, $RBUAD^k_m \geq f(min_1, min_2, \ldots, min_m)$.

Thus, $RBUAD_m = f(min_1, min_2, \ldots, min_m)$ gives a lower bound of the BUAD for the subgroup category m. □

The lower bound of the BUAD is a conservative assumption of the actual value of the BUAD. We can use this lower bound as our termination condition for our proposed algorithm.

To summarize the advantages of using the RBUAD concept are twofold (1) instead of searching all points on the boundary of $C(x,r)$ to find the optimal y we only need to find the points x_n that represent the intersections with the lines (x, q_i) and (2) instead of computing a separate $BUAD^i_m$ for every subgroup of size m we only need to evaluate once the aggregate function to obtain $RBUAD_m$.

In our algorithm, the structure of a subgroup, *SGD*, is defined as follows.

Definition 4.6. (SGD) The structure of a subgroup element *SG* contains the following attributes.

- m: the size of the subgroup, i.e., the number of users in the subgroup *SG*.
- $sgList$: the set of query points $q_1, q_2, ..., q_m$ that comprises *SG*.
- o: a data object that produces the best aggregate distance so far for *SG*.
- $sgDist$: the aggregate distance to o from $sgList$, i.e., $f(\{|q-o| : q \in sgList\})$.
- $gDist$: the aggregate distance to o for the group G, i.e., $f(\{|q-o| : q \in G\})$.
- $status$: the status of the subgroup *SG* is $temporary$ if there can be another object o' $(\neq o)$ for which $f(\{|q-o'| : q \in sgList\}) < f(\{|q-o| : q \in sgList\})$, and the status is $final$ if no such o' exists.

Notation	Meaning
G	A given set of n query points $\{q_1, q_2, \ldots, q_n\}$
$Dist(q,p)$	Euclidean distance between two points q and p
$MinDist(q,p)$	Minimum Euclidean distance between a point q and a point or an object (e.g., rectangle, circle) p
L_i	An ordered list of k items of type *SGD* with subgroup size i, where the elements are stored in order of their minimum aggregate distance to any data object
$C(x,r)$	A circular region centered at a point x with a radius r where the locations of all objects are known
$BestAggDist[i]$	k^{th} smallest aggregate distance of subgroup size i with respect to all retrieved data objects inside $C(x,r)$
$status[i]$	$status[i]$ is $temporary$ if the computation for the best subgroups of size i is not complete; otherwise $status[i]$ is $final$

Table II: Notations

4.3. Algorithm

Algorithm 1 shows the steps of k-BEST-SUBGROUPS-NN for finding the k best subgroups. The algorithm takes the following parameters as input: a set G of n query points representing user locations, a number n' denoting the minimum subgroup size, a number k representing the number of best subgroups that needs to be found from each category i for $n' \leq i \leq n$, and a function f representing the required aggregate function (SUM, MAX, or MIN).

We assume that the data objects are indexed using an *R*-tree [Beckmann et al. 1990] in the database. Our algorithm follows the best first (BF) search technique to incrementally access data objects (as discussed in the related work) and to find the results for a k-BEST-SUBGROUPS-NN query. We summarize the common notations used in this section in Table II.

The algorithm starts the search from the centroid x of the given set G of n query points $\{q_1, q_2, \ldots, q_n\}$. Since data objects are hierarchically organized using an *R*-tree, the algorithm first encounters the root node and inserts the node into a priority queue Q_p. The elements of Q_p are stored in order of their minimum distance from x.

In each iteration the algorithm pops an element p from Q_p, where p can be a data object or an *R*-tree node. If p is a data object, the algorithm computes the best subgroups and their aggregate distance to p (Lines 1.12-1.28). The computation of the best subgroups follows the same procedure described in Section 4.1. For a subgroup of size i, if the best aggregate distance, $aggDist$, in this step is smaller than the kth smallest aggregate distance computed so far, i.e., $BestAggDist[i]$, the corresponding subgroup is inserted into an ordered list L_i as the candidate answer. If p is an *R*-tree

ALGORITHM 1: k-BEST-SUBGROUPS-NN (R, G, n', k, f)

Input : An R-tree index R of all data objects, A set of query points G=$\{q_1, q_2, \ldots, q_n\}$, the number of required subgroups k from each category, the minimum number n' of users in a subgroup, and an aggregate function f (SUM, MAX, or MIN).

Output: $L = \{L_{n'}, L_{n'+1}, \ldots, L_n\}$,where L_i is an ordered list of k items of data type SGD with subgroup size i for $n' \leq i \leq n$.

```
1.1  Initialize {L_n', L_n'+1, ..., L_n} to {Ø, Ø, ..., Ø};
1.2  Initialize L to {L_n', L_n'+1, ..., L_n};
1.3  Initialize a priority queue Q_p;
1.4  x ← Calculate centroid of G;
1.5  BestAggDist[1..n − n' + 1] ← {∞};
1.6  status[1..n − n' + 1] ← {temporary};
1.7  continue ← true;
1.8  Enqueue(Q_p, R− > root, 0);
1.9  while Q_p is not empty and continue = true do
1.10     {p, MinDist(x, p)} ← Dequeue(Q_p);
1.11     if TERMINATE(G, n', x, p, f, BestAggDist, status) = exit then
1.12         continue ← false;
1.13     else if p is a data point then
1.14         Initialize a queue Q;
1.15         for each q_i in G do
1.16             Enqueue(Q, q_i, dist(q_i, p));
1.17             gDist ← gDist + dist(q_i, p);
1.18         qElem ← null;
1.19         i ← 0;
1.20         while Q is not empty do
1.21             qElem ← Dequeue(Q);
1.22             sgList ← getQueryPoint(qElem);
1.23             aggDist ← f(aggDist, qElem);
1.24             i ← i+1;
1.25             if i ≥ n' and BestAggDist[i] > aggDist then
1.26                 sg_i ← new SGD(i, sgList, p, aggDist, gDist);
1.27                 L_i ← L_i ∪ sg_i;
1.28                 UpdateBestDist(BestAggDist[i], L_i);
1.29
1.30     else
1.31         for each child node p_c of p in R do
1.32             if MinAggregateDist(p_c, G, n') < BestAggDist[n − n' + 1] then
1.33                 Enqueue(Q_p, p_c, MinDist(x, p_c));
1.34
1.35 return L;
```

node, then it retrieves its child nodes and enqueues them into Q_p if they do not satisfy the termination conditions, i.e., if there is a possibility that a child node may contain an answer (Lines 1.30-1.34). In this case, if the minimum aggregate distance to a child node p_c for the minimum subgroup size n' is greater than the kth smallest aggregate distance computed so far for the maximum subgroup size n, we can safely discard p_c. The above process continues as long as Q_p is not empty and there may exist other potential answers.

Algorithm 1 calls Algorithm 2 to check the termination conditions for the search. Algorithm 2 returns *continue* when the search needs to continue as there is a possibility that other non-retrieved objects can be the candidates for the answer set. On the other hand, Algorithm 2 returns *exit* when optimal subgroups are confirmed, i.e., all the objects that could possibly be included in the answer

ALGORITHM 2: TERMINATE (G,n',x,p,f,*BestAggDist*,*status*)

Input : A set of query points $G=\{q_1, q_2, \ldots, q_n\}$, the minimum subgroup size n', the centroid x of G, a data point or an MBR p, and an aggregate function f (SUM, MAX, or MIN).
Output: *continue* or *exit* the search.

```
2.1  lb[1..n − n′ + 1] ← {0};
2.2  r ← MinDist(x, p)
2.3  if C(x, r) does not contain G then
2.4  |   return continue;
2.5  else
2.6  |   for each q_i in G do
2.7  |   |   mq_i ← MinDist(q_i, C(x, r));
2.8  |   |   Add (q_i, mq_i) to the sorted list MQ;
2.9  |   i ← n′;
2.10 |   while i ≤ n do
2.11 |   |   lb[i − n′ + 1] ← f^i_{j=1}(mq_j);
2.12 |   |   if lb[i − n′ + 1] > BestAggDist[i − n′ + 1] then
2.13 |   |   |   status[i − n′ + 1] ← final;
2.14 |   |   i ← i+1;
2.15 |   if CheckAllStatus(status)=final then
2.16 |   |   return exit;
2.17 |   else
2.18 |   |   return continue;
2.19 |
2.20
```

set have been retrieved already. Algorithm 2 uses the lower bound defined in Lemma 4.5 to decide on the termination conditions. It uses the aggregate minimum distances from the query points to the boundary of the known region $C(x,r)$ as lower bounds, lbs, for different group sizes, and then compares lbs with the already computed *BestAggDist*s. If an lbs of all subgroup sizes are greater than the corresponding *BestAggDist*s then there is no possibility that the objects outside $C(x,r)$ can be included in the answer set, thus the algorithm returns *exit*; otherwise the algorithm returns *continue*.

The termination condition of the above algorithm is set according to the relaxed lower bound defined in Lemma 4.5. To reduce the number of nodes accessed by the above algorithm, we can find the optimal BUAD for a subgroup and then use this as our termination condition. In Appendix A.1, we show how to find the optimal BUAD for a subgroup.

It is important to note that our incremental approach can be easily extended for other variants such as the constrained consensus query, where constraints such as different priorities to different users in the group are assigned to prioritize some users' inclusion over the others in subgroup formation. Thus in such a case, we need to to consider both the priorities of the query points, and the distances between query points and data points. A way to extend our incremental approach for the constraint consensus query will be as follows: i) compute the distances from all query points to each retrieved object and store (query, priority, distance) tuples in the priority queue Q in the order of their priority first and then with their minimum distance, ii) compute the best subgroups as outlined in Steps 1 and 2 of our approach, and iii) compute the relaxed lower bound terminating condition based on the imposed constraints. For computing the BUAD for a subgroup size of m (by using Lemma 4.4), instead of considering m minimum distances from query points to the boundary of the known region, we need to include distances of query points to the boundary of the known region on the basis of priorities of the users. For example, if a member has the highest priority, the distance from that query point to the boundary of the known region must be included in BUAD computation. Then, the remaining $m-1$ distances are taken based on the minimum distances from query points to the

boundary of the known region (assuming these $m-1$ query points have same priority). Since the priority queue and the terminating condition are computed based on the priorities of query points and minimum distances, the algorithm continues retrieving objects until it finds the result satisfying the priority constraints. The above process will enable us to extend our incremental relaxed lower bound based approach (INC-R-LB) for the constraint consensus query.

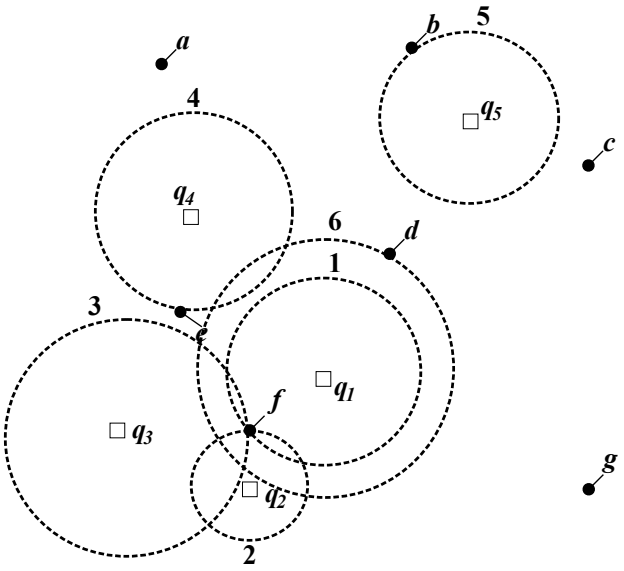

Fig. 6: Multi-query approach for evaluating consensus query. Data retrieval in 6 steps are shown using 6 circles.

4.4. An Alternative Approach

The above incremental approach finds the result in a single traversal in the database as it uses the centroid of the group as the starting point of the search. Another popular approach for GNN query processing is to consider multiple query points simultaneously and expand the search space for every query point [Papadias et al. 2004]. Similarly, to process the k-BEST-SUBGROUPS-NN query, we devise an alternative approach is to incrementally expand the search space for each query point of the group, and accesses MBRs/data points in order of their minimum distances. We call this method the multi-query (MQ) approach. In MQ, the nearest data point of q_1 is first retrieved, and then optimal subgroups of each allowable subgroup sizes (i.e., greater or equal to the minimum specified subgroup size) with respect to the retrieved data point are computed. Likewise, the nearest data points of q_2, q_3, ..., q_n are progressively retrieved and the optimal subgroups of each allowable subgroup size are updated with respect to these retrieved data points. In the next step, the second nearest data points of q_1, q_2, ..., q_n are retrieved one by one and the optimal subgroups are updated with respect to these retrieved data points. This process continues until we discover all optimal subgroups and their corresponding nearest data points.

Figure 6 shows the running example for query evaluation using the MQ approach. In the first step, the nearest data point f of q_1 is retrieved, and then the optimal subgroups of each allowable subgroup sizes (here 3, 4, and 5) with respect to f are computed (please refer to Section 4.1 for the detailed procedure of the optimal subgroup computation). In Step 2 and Step 3, f is retrieved as the nearest data point of both q_2 and q_3. Since we have already computed optimal subgroups with respect to f, we do not need any further computation in these two steps. In the subsequent steps,

e and b are retrieved as the nearest data points of q_4 (Step 4) and q_5 (Step 5), respectively. At this stage the first NNs for all query points are already considered in optimal subgroups computation. In the next iteration, d is first retrieved as the second nearest neighbor of q_1 as Step 6. The above data retrieval process continues until the result is found.

Instead of a single circular region (a disc) as our search space, the MQ search space is a union of multiple discs centered at different query points. The termination condition of the above process is formulated in the same was as the termination condition of the relaxed lower bound method as used in Algorithm 1. That is, we use the aggregate distance of the Euclidian distances between each query point and the last retrieved data point from that query point.

4.5. Discussion

In summary, we propose an incremental expansion method to evaluate the k-BEST-SUBGROUPS-NN query, which forms the building block for processing a consensus query. Depending upon the pruning strategies, the incremental expansion method can have two variants based on the exact BUAD and a relaxed BUAD. The incremental method that uses the tight lower bound for pruning the search space is called an incremental tight lower bound (INC-T-LB) approach. The tight lower bound for the BUAD incurs extra computational overheads. Thus, we derive a relaxed BUAD. We have presented that a relaxed lower bound for the BUAD can be used to prune the search space with reduced computational overhead. We name this method an incremental relaxed lower bound (INC-R-LB) approach.

The proposed incremental approaches find the result in a single traversal in the database, on the other hand, our alternative approach, the multiple query (MQ) approach, requires multiple independent searches in the database for the results.

The disadvantages of this MQ method are: (i) it may need to access the same node multiple times, (ii) the distance to an MBR needs to be calculated for each query point, (iii) the search region is not a regular shape and thus it is hard to compute the optimal BUAD.

We analyze the cost of our proposed methods with the competitive methods *Combinatorial (COMB)* and FANN in Appendix A.2.

To provide a better insight into performance comparison, we compare these algorithms through experimental studies using a wider range of parameters in a realistic location-based application setting.

Parameter	Range	Default
Data set	Real, Synthetic	Real
Real dataset size	62.5K	62.5K
Synthetic dataset size	5K, 10K, 20K, 40K	-
Query Point Distribution	Uniform (U), Zipfian (Z)	U
Group size (n)	8, 16, 32, 64	32
Min subgroup size (in % of n)	60%, 70%, 80%, 90%	60%
Area of the query space	1%, 2%, 4%, 8%, 16%	8%
k	1, 2, 4, 8, 16, 32	4

Table III: Experiment Setup

5. EXPERIMENTAL EVALUATION

We evaluate our two incremental methods, INC-R-LB and INC-T-LB, and the multi-query (MQ) approach, and compare their performances. We also compare our proposed techniques with the combinatorial (COMB) approach and the flexible aggregate nearest neighbor (FANN) approach [Li et al. 2011]. In the combinatorial approach, we need to first enumerate all possible subgroups, and then we apply the best known minimum bounding method (MBM) [Papadias et al. 2004] to find

the nearest data points for each of these subgroups. Finally, we rank the results of each subgroup category and take the top-k subgroups as our results for the k-BEST-SUBGROUPS-NN query. Since FANN is applicable for a fixed subgroup size (category), we independently execute the FANN query for each subgroup category using their proposed R-tree based algorithm and combined the results.

We vary different parameters: the group size (n), the minimum subgroup size, the area of the group/query space (the rectangular area that encloses all the query points), the number of required nearest data point (k), and the dataset size in different experiments. Table III summarizes the values used for each parameter in our experiments and their default values. We use the query CPU processing times and I/O costs (number of disk pages accessed) as the efficiency measures of the algorithms.

We use both real and synthetic datasets for our experiments. The real dataset contains 62,556 postal addresses from California. We generated synthetic datasets with uniform and Zipfian distributions, representing a wide range of real scenarios. For both distributions, we vary the dataset size as 5000, 10,000, 15,000, and 20,000 point locations. For all datasets, we normalize the data space into a span of $10,000 \times 10,000$ square units. In a query space, the query points are generated using both Uniform and Zipfian distribution. Note that the Zipfian query distribution ensures skewed distribution of users in a group.

In our experiments, data points are indexed using an R^*-tree [Beckmann et al. 1990]. We used a page size of 1KB and a node capacity of 50 entries for the R^*-tree. For each set of experiments, we evaluate all algorithms for 100 groups of users and present the average experimental results. To generate 100 different groups, we first randomly generate 100 points in the data space. Then, we define a rectangular query space centered at each generated point. Finally, the required query points in a group are generated inside the rectangular query space.

We run the experiments on a PC with a Core i5-2430M CPU @ 2.40 GHz and 4 GByte RAM.

5.1. Comparison with Baseline Methods

We first present the scalability test by comparing our three approaches, INC-R-TB, INC-T-TB, and MQ, with two baseline methods COMB and FANN. In this set of experiments, we vary the group size using 8, 16, and 24, as the COMB approach does not scale for higher group sizes (e.g., 32, 64). We measure the query processing time and I/O cost for aggregate functions SUM, MAX, and MIN.

Figures 7(a)-(b), (c)-(d), and (e)-(f) show the query processing time and I/O of all five methods for the aggregate functions SUM, MAX, and MIN, respectively. We can see that our proposed method INC-R-LB is the best performer in terms of processing time, while INC-T-LB is the best performer in terms of I/O cost. Note that both of our approaches INC-R-LB and ICN-T-LB developed based on relaxed bound and tight bound, respectively, produce accurate results. However, relaxed bound based approach, INC-R-LB, needs to retrieve more data from the database to accurately answer the query. The experimental results also show that INC-T-LB requires slightly lower number of I/O cost than that of INC-R-LB at the cost of higher processing time. For example, for a group size of 16 for the aggregate function SUM, INC-T-LB incurs 2 times higher processing time with only 4% less number of I/O cost (i.e., retrieving roughly 4% less data points) than that of INC-R-LB. Thus, INC-R-LB is the preferred method over INC-T-LB if we consider the tradeoff between the processing time and I/O costs. Hence, we only consider INC-R-LB in the detailed comparative analysis. Note that, in the case of INC-T-LB, we avoid the generation of exponential number of subgroups and use an approximation based approach that discretizes the boundary of the known circular region and consider every point of the boundary to find the BUADs of all subgroup categories. In order to discretize the circle, we take the perimeter of the circle as the length of the boundary region and take every integer number as a boundary point.

Figures for SUM also show that COMB does not scale even for a moderate group size of 24. For an increase of group size from 8 to 24, the processing time and I/O cost increase by more than 3 orders of magnitude. In the case of FANN, for an increase of group size from 8 to 24, the processing time and I/O cost increase by more than 6 times. On the other hand, both of our

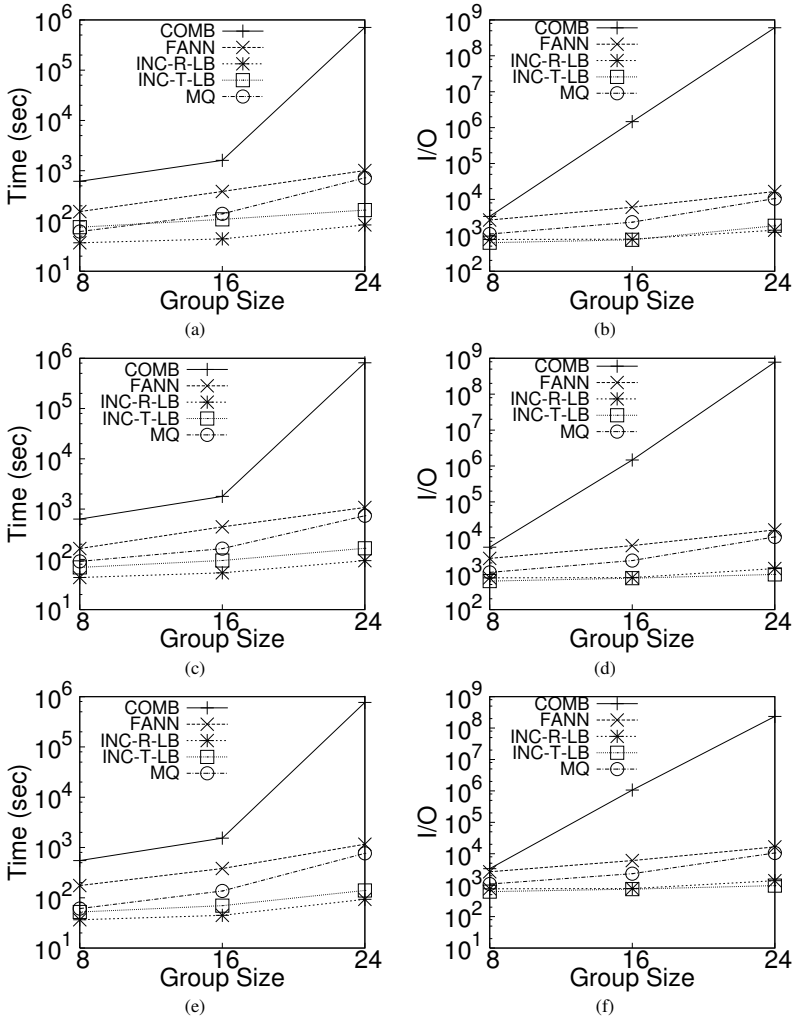

Fig. 7: Scalability for aggregate functions SUM (a-b), MAX (c-d), and MIN (e-f)

incremental approaches, INC-R-LB and INC-T-LB outperform COMB and FANN significantly. For example, for a group size of 24, both INC-R-LB and INC-T-LB outperform COMB approach by at least 2 orders of magnitude in processing time and 4 orders of magnitude in I/Os. For the same group size, our incremental approaches outperform FANN by an order of magnitude both in terms of processing cost and I/Os. We can see from the figure that our alternative MQ approach also outperform COMB and FANN significantly. Note that for aggregate functions MAX and MIN, all competitive methods exhibit similar behavior to that shown for SUM. Since the COMB approach is not practical due to huge computational and I/O overhead, we omit comparing this approach with ours in the rest of the comparative analysis.

Note that as described in Section 4.1, in our incremental approach we start the BF search from the centroid of the query points. An alternative starting point of the search could be a random starting point in the query space. In the random starting point based approach, we randomly select a point

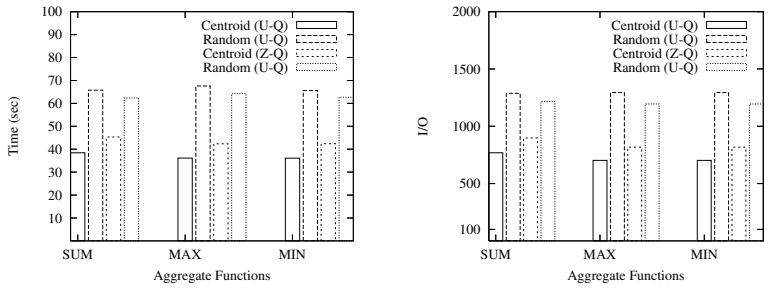

Fig. 8: Random starting point vs. centroid based starting point in our INC-R-LB method for aggregate functions SUM, MAX, and MIN.

inside the query space and incrementally retrieve the data points in increasing order of their distances from the starting point. For all default values of parameters as stated in Table III, the centroid based approach requires much less query processing time and I/O overhead than the random starting point based approach for all three aggregate functions (Figure 8). For aggregate function SUM with Uniform distribution of query points, the centroid based approach requires 71% less processing time and 67% less I/Os than the random starting point based approach. Also, for aggregate function SUM with Zipfian distribution of query points, the centroid based approach requires 28% less processing time and 35% less I/Os than the random starting point based approach. For aggregate function MAX with Uniform distribution of query points, the random starting point based approach takes 87% and 84% more time and I/Os, respectively, than the centroid based approach. On the other hand, for Zipfian distribution of query points, the centroid based approach requires 51% and 46% less time and I/Os, respectively than the random starting point based approach. For aggregate function MIN, the results show similar trends to that of aggregate function MAX. This empirical study validates our choice of centroid as the starting point over a random starting point.

5.2. Effect of Varying the Query Space

In these experiments, we vary the query area, i.e., the area to which the group members are confined to, as 1%, 2%, 4%, 8%, and 16% of the total data space. Figures 9 show the required time and I/O cost for processing k-BEST-SUBGROUPS-NN query using INC-R-LB, MQ, and FANN for aggregate functions SUM (a-b), MAX (c-d), and MIN (e-f). We conduct experiments for both Uniform (U) and Zipfian (Z) distributions of query locations. Figures 9 (a)-(b) show that for both U and Z query distributions, INC-R-LB is 5 and 10 times faster than the MQ and FANN methods, respectively. INC-R-LB requires 5 and 10 times less I/O cost than that of MQ and FANN methods, respectively. We also see that the processing time and I/O cost for all methods increase with the increase of the query space, which is expected because for a larger query space all algorithms require to access more database objects.

For aggregate functions MAX (Figures 9 (c)-(d)) and MIN (Figures 9 (e)-(f)), we see a similar behavior as SUM.

5.3. Effect of Varying the Group Size

In this set of experiments, we vary the group size as 8, 16, 32 and 64, and measure the required processing time and I/O cost in INC-R-LB, MQ, and FANN methods for both Uniform (U) and Zipfian (Z) query distributions. Figure 10 shows the required processing time and I/O cost for SUM (a-b), MAX (c-d), and MIN (e-f). We can see that the processing time and I/O cost slightly increase with the increase of group size for INC-R-LB. On the other hand, in case of MQ and FANN, both the processing time and I/O cost significantly increase with the increase of group size. Thus, we

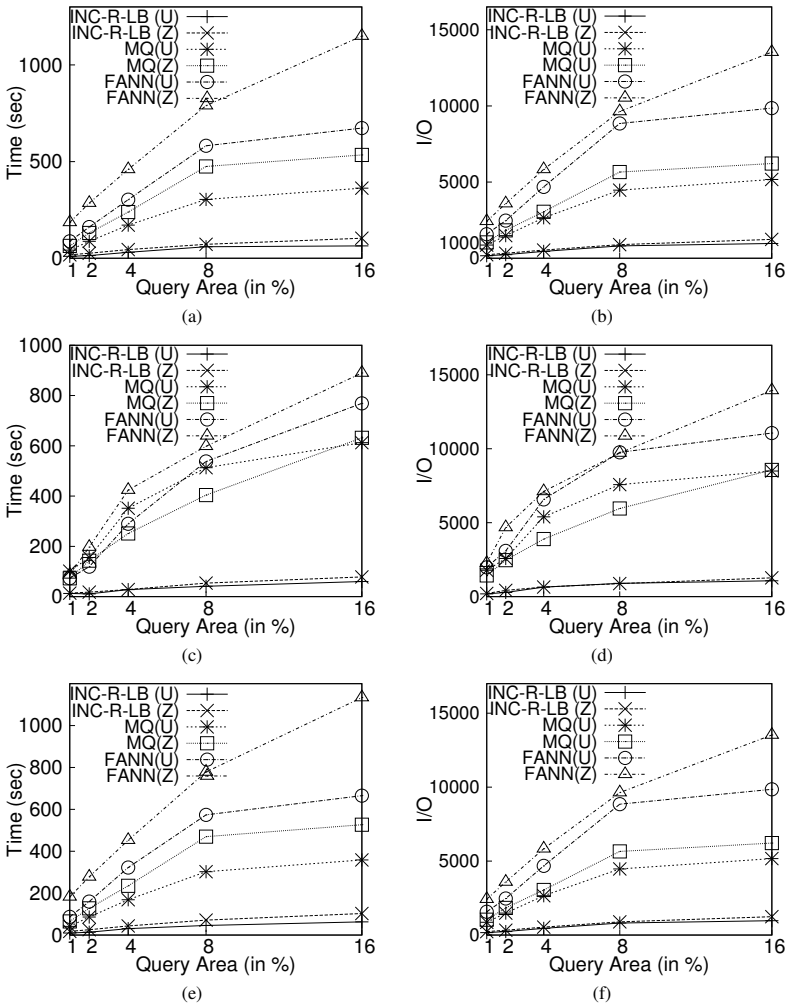

Fig. 9: The effect of varying the query space for SUM (a-b), MAX (c-d), and MIN (e-f)

observe that our approach INC-R-LB outperforms MQ and FANN in a greater margin for a larger group size. Figure 10 (a) shows that INC-R-LB is on average 4 times faster than MQ and 5 times faster than FANN for a group of size 8, and 15 times faster than MQ and 30 times faster than FANN for a group of size 64. Similarly, Figure 10 (b) shows that INC-R-LB requires on the average 1.5 and 10 times less I/O cost than MQ for a group size 8 and 64, respectively. Figure 10 (b) also shows that INC-R-LB requires on an average 4 and 31 times less I/O cost than FANN for a group size 8 and 64, respectively.

Figure 10 (c)-(d) and (e)-(f) show the results for aggregate functions MAX and MIN, respectively, which depict similar trends as SUM.

5.4. Effect of Varying the Minimum Subgroup Size

In these experiments, we vary the minimum subgroup size (n'), i.e., the minimum number of users required to form a valid group of 60%, 70%, 80%, or 90% of users from the group. Figure 11 shows

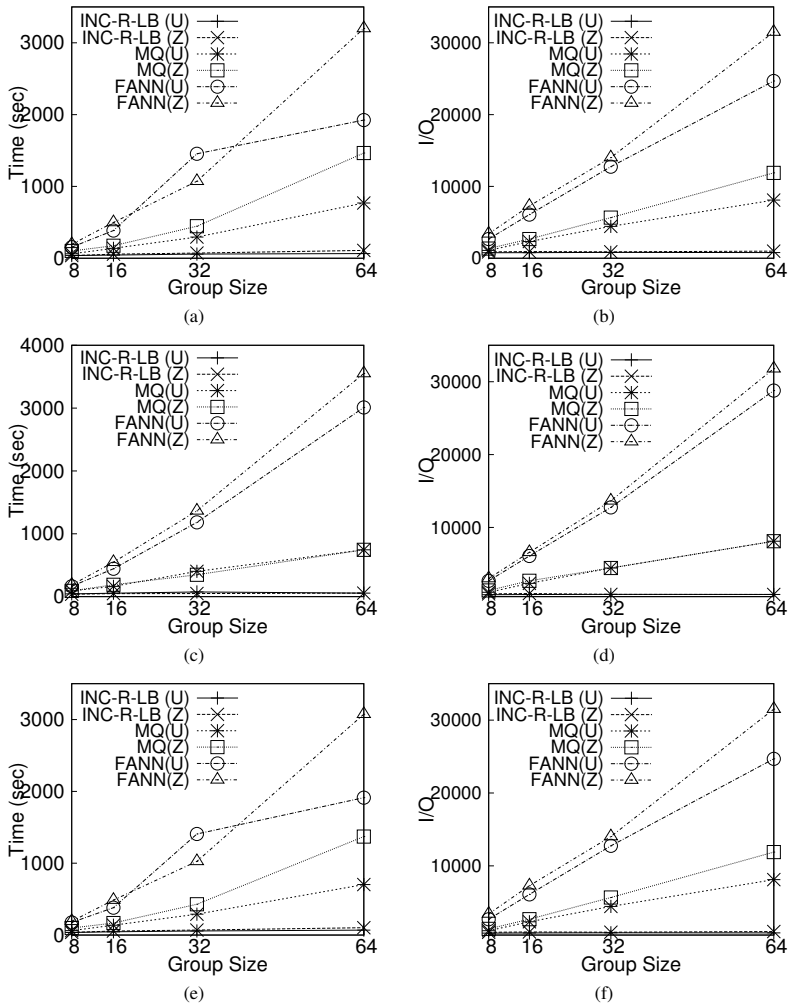

Fig. 10: The effect of varying the group size for SUM (a-b), MAX (c-d), and MIN (e-f)

that for aggregate function SUM the query processing time decreases with the increase of minimum subgroup size. Since the total number subgroup decreases with the increase of n', the computational overhead decreases. On the other hand, we can see that the I/O costs almost remain constant for varying the minimum subgroup size. Since, in general, the optimal subgroup of a larger subgroup size is a superset of the optimal subgroup of a smaller subgroup size, the number of data points needed to find the optimal subgroup of a larger subgroup category is sufficient to find the optimal subgroup of a smaller subgroup category. Figure 11 also shows that INC-R-LB outperforms MQ and FANN significantly for all cases. The results for aggregate functions MAX and MIN show similar trends as SUM (not shown).

5.5. Effect of Varying k

In this set of experiments, we vary the value of k using 1, 2, 4, 8, 16, and 32 for processing k-BEST-SUBGROUPS-NN query. We observe that both the processing cost and I/Os increase with the

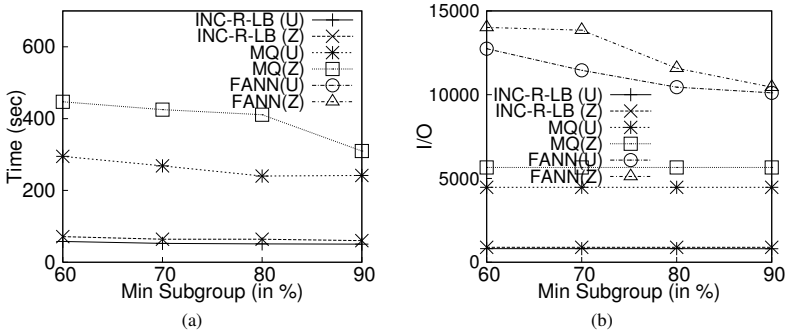

Fig. 11: The effect of varying n' for SUM

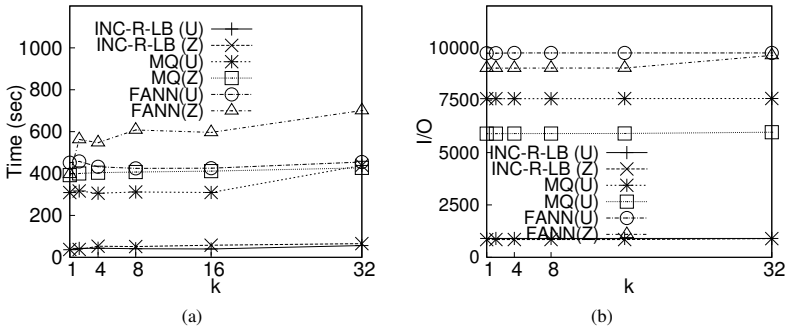

Fig. 12: The effect of varying k for SUM

increase of the values of k for all aggregate functions. The results show that INC-R-LB significantly outperforms MQ and FANN in all evaluations for aggregate functions SUM(Figure 12).

We see that the I/O costs also slightly increase with the increase of the values of k. Figures show that INC-R-LB significantly outperforms MQ and FANN in all evaluations. The results for aggregate functions MAX and MIN depict similar trends as SUM (not shown).

5.6. Effect of Varying the Dataset Size

Figure 13 shows the required processing time and I/O cost in INC-R-LB, MQ, and FANN methods for varying dataset size for aggregate function SUM. In this set of experiments, we vary the dataset size using 5K, 10K, 20K, and 40K for both Uniform (a-b) and Zipfian (c-d) data distributions. For both uniform and Zipfian distributions, the processing time and I/O cost increase with the increase of the dataset size. Figures show that INC-R-LB outperforms both MQ and FANN in a greater margin for a larger dataset size. The experimental results also show that with the increase of dataset size, query processing time and I/O cost increase linearly in our approach. For example, if we double the dataset size from 20K to 40K, the query processing time increases by 54%. Thus, our approach is scalable for a large dataset

We omit the graphs for aggregate functions MAX and MIN, as the results are similar to SUM.

6. CONCLUSION

We have proposed a new type of query, called the consensus query, that allows a group of users to share their locations and collaboratively find a solution that suits most of the members in the group. To facilitate the consensus queries, we have developed efficient algorithms for the LSP to

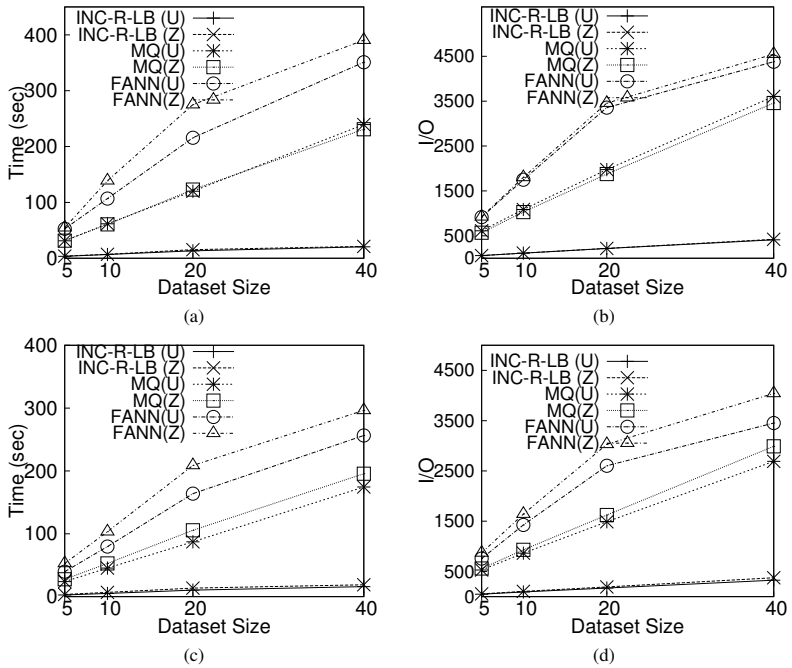

Fig. 13: The effect of varying dataset size in U (a-b) and Z (c-d) distributions for SUM

find k best subgroups of all allowable subgroup sizes (i.e., greater or equal to the minimum specified subgroup size) that minimize the aggregate travel distance to data points. Our proposed algorithm for finding the optimum subgroups and the corresponding nearest data points significantly outperform all competitive methods in terms of both query processing time and I/O costs. Experimental results show that the combinatorial approach does not scale beyond the group size of 24. On the other hand, our proposed incremental method is scalable for a large group size. Experimental results also reveal that our incremental approach performs at least 2 times better than that of the multi-query approach and 5 times better than FANN approach and the incremental approach outperforms both approaches by 13 and 30 times, respectively, for a larger group size of 64.

In the future, we will extend our model to process the consensus query in road networks. Another interesting future research direction is to consider a user location as a region instead of a point which is desirable from the standpoint of privacy.

REFERENCES

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An efficient and robust access method for points and rectangles. In *SIGMOD*. 322–331.

Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *SIGMOD*. 373–384.

Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial Keyword Query Processing: An Experimental Evaluation. *PVLDB* 6, 3 (2013), 217–228.

Chi-Yin Chow, Jie Bao, and Mohamed F. Mokbel. 2010. Towards location-based social networking services. In *2nd ACM SIGSPATIAL International Workshop on LBSN*. 31–38.

Ke Deng, Shazia Wasim Sadiq, Xiaofang Zhou, Hu Xu, Gabriel Pui Cheong Fung, and Yansheng Lu. 2012. On group nearest group query processing. *IEEE TKDE* 24, 2 (2012), 295–308.

Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword Search on Spatial Databases. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*. 656–665.

Sébastien Ferré and Alice Hermann. 2011. Semantic search: reconciling expressive querying and exploratory search. In *ISWC*. 177–192.

FourSquare. 2011. (2011). http://foursquare.com/.

Mike Gartrell, Xinyu Xing, Qin Lv, Aaron Beach, Richard Han, Shivakant Mishra, and Karim Seada. 2010. Enhancing group recommendation by incorporating social relationship interactions. In *GROUP*. 97–106.

Gowalla. 2011. *http://gowalla.com/*.

Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*. 47–57.

Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, and Lars Kulik. 2013. Group Trip Planning Queries in Spatial Databases. In *SSTD*. 259–276.

Gisli R. Hjaltason and Hanan Samet. 1995. Ranking in spatial databases. In *SSD*. 83–95.

iGroups. 2011. *http://itunes.apple.com/us/app/igroups*.

Anthony Jameson and Barry Smyth. 2007. Recommendation to groups. In *The Adaptive Web*. 596–627.

Christian S. Jensen, Jan Kolářvr, Torben Bach Pedersen, and Igor Timko. 2003. Nearest neighbor queries in road networks. In *ACM GIS*. 1–8.

Anastasios Kementsietsidis, Frank Neven, Dieter Van de Craen, and Stijn Vansummeren. 2008. Scalable multi-query optimization for exploratory queries over federated scientific databases. *PVLDB* 1, 1 (2008), 16–27.

Mohammad Kolahdouzan and Cyrus Shahabi. 2004. Voronoi-based K nearest neighbor search for spatial network databases. In *VLDB*. 840–851.

Yang Li, Feifei Li, Ke Yi, Bin Yao, and Min Wang. 2011. Flexible aggregate similarity search. In *SIGMOD*. 1009–1020.

Loopt. 2011. *http://www.loopt.com/*.

Sarah Masud, Farhana Murtaza Choudhury, Mohammed Eunus Ali, and Sarana Nutanong. 2013. Maximum visibility queries in spatial databases. In *ICDE*. 637–648.

Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. 2004. Group nearest neighbor queries. In *ICDE*. 301–310.

Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.* 30, 2 (2005), 529–576.

Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. 2003. Query processing in spatial network databases. In *VLDB*. 802–813.

João B. Rocha-Junior and Kjetil Nørvåg. 2012. Top-k spatial keyword queries on road networks. In *EDBT*. 168–179.

Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. 1995. Nearest neighbor queries. In *SIGMOD*. 71–79.

Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. In *WWW*. 851–860.

Thomas Seidl and Hans-Peter Kriegel. 1998. Optimal multi-step k-nearest neighbor search. *SIGMOD Rec.* 27 (1998), 154–165. Issue 2.

Jing Shan, Donghui Zhang, and Betty Salzberg. 2003. On spatial-range closest-pair query. In *SSTD*. 252–269.

Ben Shneiderman. 1994. Dynamic queries for visual information seeking. *IEEE Softw.* 11, 6 (Nov. 1994), 70–77.

Xiance Si, Edward Y. Chang, Zoltán Gyöngyi, and Maosong Sun. 2010. Confucius and its intelligent disciples: integrating social with search. *PVLDB* 3, 2 (2010), 1505–1516.

E Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). *In New Results and New Trends in Computer Science* 555, 1 (1991), 359–370.

Chenyi Xia, Hongjun Lu, Beng Chin Ooi, and Jin Hu. 2004. Gorder: An efficient method for KNN join processing. In *VLDB*. 756–767.

De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. 2012. On socio-spatial group query for location-based social networks. In *KDD*. 949–957.

Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. 2005. Aggregate nearest neighbor queries in road networks. *IEEE TKDE* 17 (2005), 820–833.

Tjalling J. Ypma. 1995. Historical development of the Newton-Raphson method. *SIAM Rev.* 37 (1995), 531–551. Issue 4.

Zhiyong Yu, Zhiwen Yu, Xingshe Zhou, and Yuichi Nakamura. 2009. Handling conditional preferences in recommender systems. In *IUI*. 407–412.

Jinzeng Zhang, Xiaofeng Meng, Xuan Zhou, and Dongqi Liu. 2012. Co-spatial Searcher: Efficient Tag-Based Collaborative Spatial Search on Geo-social Network.. In *DASFAA (1)*. 560–575.

Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

A. APPENDIX

A.1. Computing the BUAD

The BUAD for SUM: For a given known region $C(x,r)$, to find the optimal BUAD for a subgroup $\{q_1, q_2, \ldots, q_m\}$ of m users, we need to identify a point y on the boundary point set ß of $C(x,r)$ that minimizes the summation of distances from all query points, $q_1, q_2, \ldots, q_m$ to ß, i.e., y minimizes $\sum_{i=1}^{m} |q_i - y|$.

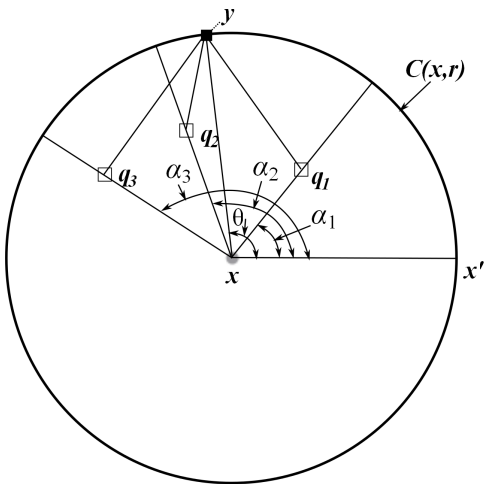

Fig. 14: The known circular region $C(x,r)$ with a center x and a radius r, three query points $\{q_1, q_2, q_3\}$.

Let θ denote the angle between $\overline{xx'}$ and $\overline{xy}$ measured counterclockwise with respect to x', i.e., $\angle x'xy = \theta$ as shown in Figure 14.

Also assume that α_i corresponds to $\angle x'xq_i$, i.e.,α_1, α_2, α_3 corresponds to the query points q_1, q_2, and q_3, respectively. In Figure 14, for $\triangle xq_1y$, according to the cosine formula of a triangle we can write the following equation.

$$|q_1 - y|^2 = |x - q_1|^2 + r^2 - 2 \cdot |x - q_1| \cdot r \cdot \cos(\theta - \alpha_1)$$

Similarly, for any query point q_i, we can write the following equation for a triangle $\triangle xq_iy$.

$$|q_i - y|^2 = |x - q_i|^2 + r^2 - 2 \cdot |x - q_i| \cdot r \cdot \cos(\theta - \alpha_i)$$

We can see that $|x - q_i|$, r, and α_i are constants for a given query point q_i and a circular region $C(x,r)$. Hence, we can simplify the above equation by replacing $|x - q_i|^2 + r^2$ with a constant A_i and $2 \cdot |x - q_i| \cdot r$ with a constant B_i.

$$|q_i - y|^2 = A_i - B_i \cdot \cos(\theta - \alpha_i)$$

Based on the above formulations, we can express our optimization function for m query points $\{q_1, q_2, \ldots, q_m\}$ as a function of θ as follows.

$$g(\theta) = \sum_{i=1}^{m} |q_i - y|$$
$$= \sum_{i=1}^{m} (\sqrt{A_i - B_i \cdot \cos(\theta - \alpha_i)})$$

If we take the first derivative of $g(\theta)$, we have the following equation.

$$\frac{dg(\theta)}{d\theta} = \sum_{i=1}^{m} \frac{B_i(\cos\alpha_i \cdot \sin\theta - \sin\alpha_i \cdot \cos\theta)}{2 \cdot \sqrt{A_i - B_i \cdot (\cos\alpha_i \cdot \cos\theta + \sin\alpha_i \cdot \sin\theta)}}$$

To find the optimal θ, we need to solve $\frac{dg(\theta)}{d\theta} = 0$.

Since the first derivative is a complex series comprising of square roots of different terms, the straightforward trigonometric solution is not possible in this case. Thus, we use Newton-Raphson [Ypma 1995] method to solve the above equation, which can be described as follows.

Given a function $g(\theta)$ and its derivative $g'(\theta)$, the Newton-Raphson method begins with an initial guess θ_0, and then finds a better approximation θ_1 by using the following formula.

$$\theta_1 = \theta_0 - \frac{g(\theta_0)}{g'(\theta_0)}$$

Then the above process is repeated until a sufficiently accurate value is found.

$$\theta_n = \theta_{n-1} - \frac{g(\theta_{n-1})}{g'(\theta_{n-1})}$$

Since we need to find a minimum of a function, and the derivative is zero at a minimum, we can find the minima by applying Newton-Raphson method to the derivative, i.e., the above iteration becomes:

$$\theta_n = \theta_{n-1} - \frac{g'(\theta_{n-1})}{g''(\theta_{n-1})}.$$

The Newton-Raphson method converges quickly to the solution if the initial starting point θ_0 is a good approximation. For this, we choose initial θ_0 as the angle of the centroid of subgroup query points at x.

By using the above method, we can compute the BUAD for any subgroup of size m. Since there are $\binom{n}{m}$ possible subgroups of size m, the BUAD for the subgroup category m, $BUAD_m$, can be computed by taking the minimum of all BUADs, i.e., $BUAD_m = \min_{i=1}^{\binom{n}{m}}(BUAD_m^i)$.

The BUAD for MAX: To find the BUAD for MAX, we need to find a point y on the boundary of $C(x,r)$ that minimizes the maximum distances for a subgroup $sg_m = \{q_1, q_2, \ldots, q_m\}$ to the boundary point set ß of $C(x,r)$. That is, find $y \in$ ß that minimizes $\max_{i=1}^{m} |q_i - y|$. This problem can be expressed as finding the minimum circle centered at any point $y \in$ ß that encloses all query points, where the radius of the circle denotes the BUAD for the subgroup sg_m.

First, we show how to find the BUAD for any subgroup of two query points, $\{q_i, q_j\}$, and then we generalize the idea for a group of m query points.

Let the minimum and maximum distances from q_i to the boundary point set ß of $C(x,r)$ be $q_i x_i$ and $q_i x_i'$, respectively. If we move along the boundary points away from x_i (in either direction), the distance from q_i to ß continuously increases and it reaches the maximum at point x_i'. Similarly, the distance from q_j to ß continuously increases as we move along the boundary points away from x_j.

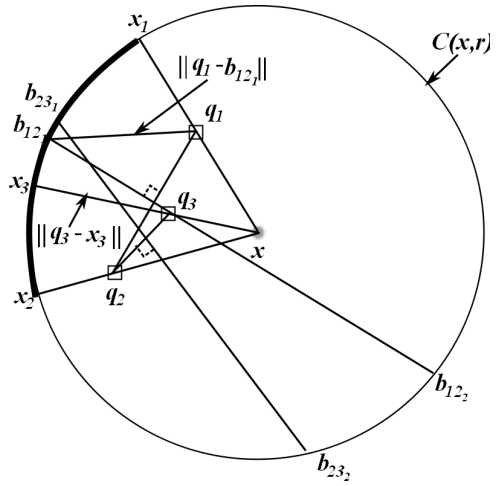

Fig. 15: The known circular region $C(x,r)$, the BUAD $|q_2 - b_{121}|$ for $\{q_1, q_2\}$, and the BUAD $|q_3 - x_3|$ for $\{q_2, q_3\}$.

Thus, the point $y \in$ ß that minimizes the maximum distances from subgroup $\{q_i, q_j\}$ to ß must be on the arc $\widehat{x_i x_j}$ bounded by the smaller angle $\angle x_i x x_j$ that ranges between 0^o and 180^o. Figure 15 shows the arc $\widehat{x_1 x_2}$ using boldface as a candidate set of points for y.

To determine a point $y \in x_i x_j$ that minimizes the maximum distances for q_i and q_j, we will first look at an example as shown in Figure 15. In this example, the point that minimizes the maximum distances for q_1 and q_2 must reside on the arc $\widehat{x_1 x_2}$. The figure shows that the bisector $\overline{b_{121} b_{122}}$ of q_1 and q_2 meets the boundary of $C(x,r)$ at two points b_{121} and b_{122}, respectively. The point b_{121} is on the arc $\widehat{x_1 x_2}$ and the aggregate distance from q_1 and q_2 to border point set ß is minimized at b_{121}. This is because: (i) the distances $|q_1 - b_{121}|$ and $|q_2 - b_{121}|$ are equal, (ii) any point that is right to b_{121} will have a greater distance from q_2 than $|q_2 - b_{121}|$, and (iii) any point that is left to b_{121} will have a greater distance from q_1 than $|q_1 - b_{121}|$. In this case, the intersection point between the bisector of two query points and the smaller arc formed by the two query points and the center minimizes the maximum distances for two query points $\{q_1, q_2\}$ to the boundary point set ß. If we consider two query points q_2 and q_3 as shown in Figure 15, the point that minimizes the maximum distances for subgroup $\{q_2, q_3\}$ must reside on the arc $\widehat{x_2 x_3}$. However, we can see that the bisector of q_2 and q_3 does not intersect $\widehat{x_2 x_3}$. In this case, the minimum distance from q_3 to ß is $|q_3 - x_3|$, and the distance $|q_2 - x_3|$ from q_2 to x_3 is smaller than $|q_3 - x_3|$. Thus the point x_3 minimizes the aggregate distances for subgroup $\{q_2, q_3\}$ for MAX. Based on the above discussion, we have the following lemma for finding the BUAD for any two query points.

LEMMA A.1. *Let q_i and q_j be two query points inside a circular region $C(x,r)$ centered at a point x with a radius r, and $|q_i - x_i|$ and $|q_j - x_j|$ be the minimum distances from q_i and q_j to the boundary point set ß of $C(x,r)$, respectively. Let $\widehat{x_i x_j}$ be the smaller arc formed by the $\angle x_i x x_j$ that ranges from 0^o to 180^o. Then a point $y \in$ ß that minimizes the maximum distances for subgroup $\{q_i, q_j\}$ can be computed as follows:*

(1) If the circle centered at x_i (x_j) with a radius $|x_i - q_i|$ ($|x_j - q_j|$) contains q_j (q_i), then $y = x_i$ (x_j).

Otherwise,

(2) y is the intersection point of the bisector $\overline{b_{ij_1} b_{ij_2}}$ of the two query points q_i,q_j and the arc $\widehat{x_i x_j}$.

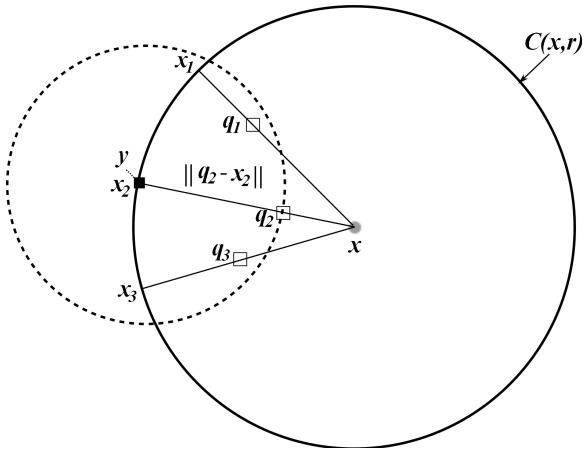

Fig. 16: The known circular region $C(x,r)$, query points $\{q_1,q_2,q_3\}$, and the BUAD $|q_2-x_2|$.

We generalize the above concept for a subgroup of any size (greater than 2). Without loss of generality, let us consider an example as shown in Figure 16. In this figure, the minimum distances from query points q_1, q_2, and q_3 to the boundary of the known region $C(x,r)$ are $|q_1-x_1|$, $|q_2-x_2|$, and $|q_3-x_3|$, respectively. Among these distances, the minimum distance $|q_2-x_2|$ of q_2 is maximized. However, the distances from q_1 and q_3 to x_2 is less than $|q_2-x_2|$, which is the minimum distance from q_2 to any boundary points of $C(x,r)$. It means that x_2 is a point that minimizes the maximum distance for three query points $\{q_1,q_2,q_3\}$, and the BUAD for this subgroup is $|q_2-x_2|$.

Let us consider a scenario, where q_1 is outside the circle centered at x_2 with a radius $|q_2-x_2|$, as shown in Figure 17. In such a case, we show that the BUAD can be computed from the bisectors of query points. In our example in Figure 17, $|b_{131}-q_1|$ is the BUAD as the circle centered at b_{131} with a radius $|b_{131}-q_1|$ contains all query points and this is the smallest possible circle centered at any boundary point of $C(x,r)$ which bounds all query points. The reason is as follows. In Figure 17, $\overline{b_{121}b_{122}}$, $\overline{b_{131}b_{132}}$, $\overline{b_{231}b_{232}}$ are the bisectors of $\overline{q_1q_2}$, $\overline{q_1q_3}$, and $\overline{q_2q_3}$, respectively. The minimum distance points from q_1, q_2, and q_3 to the boundary of $C(x,r)$ are x_1, x_2, and x_3, respectively. In this case, a point that will minimize the maximum distances among these three query points will be a point on the arc $\widehat{x_1x_3}$. Since the bisecting point b_{131} minimizes the maximum distances from q_1 and q_3 to the boundary point set of $C(x,r)$, i.e., $|q_1-b_{131}|$, and the distance from q_2 to b_{131} is less than $|q_1-b_{131}|$, the BUAD is $|q_1-b_{131}|$ for subgroup $\{q_1,q_2,q_3\}$.

Based on the above observation, we can have the following lemma.

LEMMA A.2. *Let $sg_m=\{q_1,q_2,\ldots,q_m\}$ denote a set of m query points, $C(x,r)$ a circular region centered at a point x with a radius r. Then the BUAD of subgroup sg_m can be computed as follows:*

(1) Let $|q_i-x_i|$ be the minimum distance from q_i to the boundary point set ß of $C(x,r)$. If $|q_j-x_j|$ is the maximum of all minimum distances $\{|q_1-x_1|,|q_2-x_2|,\ldots,|q_m-x_m|\}$ and the circle centered at x_j with a radius $|q_j-x_j|$ contains all the query points, then $|q_j-x_j|$ is the BUAD for subgroup sg_m; Otherwise,

(2) Let b_{ij} be the bisector of q_i and q_j, and b_{ij_1} and b_{ij_2} be the two points where the bisector b_{ij} meets the boundary point set ß of $C(x,r)$. Let B be the set of distances $|b_{ij_l}-q_i|$, where $1\leq i,j\leq m$ and $1\leq l\leq 2$, such that the circle centered at b_{ij_l} with a radius $|b_{ij_l}-q_i|$ contains all query points. Then, the distance $|b_{rtl}-q_r|\in B$ is the BUAD where $|b_{ij_l}-q_i|\geq|b_{rtl}-q_r|$ for any $|b_{ij_l}-q_i|\in B\setminus\{|b_{rtl}-q_r|\}$.

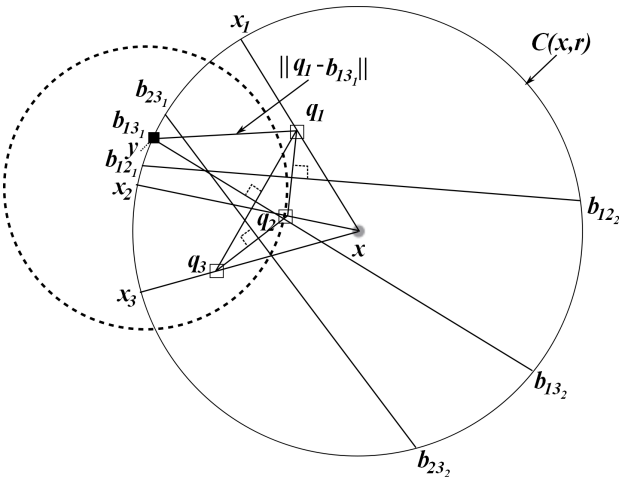

Fig. 17: The known circular region $C(x,r)$, query points $\{q_1,q_2,q_3\}$, and the BUAD $|q_1-b_{131}|$.

PROOF.

We will first prove the first part of the lemma. Let $|q_j-x_j|$ be the maximum of all minimum distances $\{|q_1-x_1|,|q_2-x_2|,\ldots,|q_m-x_m|\}$ from m query points to the boundary point set ß. Now, if we draw a circle centered at x_j with a radius $|q_j-x_j|$ and the circle contains all other query points, then the distances from all those query points to x_j will be smaller than the minimum distance $|q_j-x_j|$ of q_j to the boundary point set ß. Since for any point $y\in ß\setminus x_j$, the distance from q_j to y will be higher than $|q_j-x_j|$, the point x_j minimizes the maximum distances for subgroup sg_m and $|q_j-x_j|$ is the BUAD.

If the BUAD cannot be computed using the above process, we rely on the bisectors of query points as described in the second part of the lemma.

Given the list of bisectors $\overline{b_{ij_1}b_{ij_2}}$, where $1<=i,j<=m$, of all pairs of query points, we will prove that the BUAD can be computed from the intersection of bisectors and the boundary point set ß of $C(x,r)$. We will show that, if the circle centered at b_{rtl} with a radius of $|b_{rtl}-q_r|$ (or $|b_{rtl}-q_t|$) is the minimum radius circle in B, then $b_{rtl}\in ß$ minimizes the maximum distances for subgroup sg_m.

Let b_{rt1} and b_{rt2} be the two intersection points of the bisector $\overline{b_{rt1}b_{rt2}}$ and the boundary point set ß of $C(x,r)$. Let us also assume that b_{rt1} and b_{rt2} meet two arcs bounded by the smaller angle $\angle q_rxq_t$ that ranges from 0^o to 180^o and the larger angle $\angle q_txq_r$ which is greater than 180^o, respectively.

Let us first consider the case when $\overline{b_{rt1}b_{rt2}}$ meets the arc $\widehat{q_rq_t}$ bounded by the smaller angle $\angle q_rxq_t$, i.e., $b_{rtl}=b_{rt1}$. In this case, since b_{rt1} minimizes the maximum distance for q_r and q_t (as shown in Lemma A.1(2)) and contains all other query points, $|b_{rtl}-q_r|$ is the BUAD. This is because, for any other point $y'\in ß\setminus b_{rt1}$, the aggregate distance from q_r and q_t will increase, as b_{rt1} minimizes the maximum distances from q_r and q_t to ß.

Let us consider the case when $\overline{b_{rt1}b_{rt2}}$ meets the arc $\widehat{q_rxq_t}$ bounded by the larger angle $\angle q_rxq_t$ which is greater than 180^o. That is , i.e., $b_{rtl}=b_{rt2}$. In this case, the point b_{rt2} does not minimize the maximum distances from q_r and q_t to boundary point set ß of $C(x,r)$. We will prove that b_{rtl} still minimizes the maximum distance for all query points. To prove this, we first identify a set of candidate points in ß that have the possibility of minimizing the maximum distance for query points.

Given $b_{rtl}-q_r$ is the smallest distance in B that contains all query points. Let $(q_r,|q_r-b_{rt2}|)$ and $(q_t,|q_r-b_{rt2}|)$ be two circles centered at q_r and q_t with radii $|q_r-b_{rt2}|$ and $|q_t-b_{rt2}|$, respectively. Any point inside the circle $(q_r,|q_r-b_{rt2}|)$ will have a smaller distance to q_r than the distance $|q_r-$

$b_{rt2}|$, and similarly any point inside the circle $(q_t, |q_t - b_{rt2}|)$ will have a smaller distance from q_t than the distance $|q_t - b_{rt2}|$. Thus the boundary point set ß of $C(x,r)$ that falls inside the intersection of two circles, $(q_r, |q_r - b_{rt2}|)$ and $(q_t, |q_r - b_{rt2}|)$, can have a smaller distance from q_r and q_t than the aggregate distance from q_r and q_t to b_{rt2}. Let the arc $\widehat{y_1y_2}$ be the set of boundary points that can have a smaller maximum distance than the distance $|q_r - b_{rt2}|$ for q_r and q_t. (Figure 18 shows an example where the arc $\widehat{y_1y_2}$ is formed by two circles $(q_2, |q_2 - b_{232}|)$ and $(q_3, |q_3 - b_{232}|)$)). Now, we need to show that no point on the arc $\widehat{y_1y_2}$ can have a smaller circle that encloses all query points of sg_m.

We will first show that the circles centered at two extreme points y_1 and y_2 of the arc $\widehat{y_1y_2}$ with a radius equal to or smaller than $|q_r - b_{rt2}|$ cannot contain all query points. We now show that the circle centered at y_1 with a radius $|y_1 - q_r|$ (or $|q_r - b_{rt2}|$) cannot include all query points. That is, there is at least one query point q_i that falls outside of the circle $(y_1, |y_1 - q_r|)$. Let us first assume that q_i is on the border of the circle $(y_1, |y_1 - q_r|)$. In that case the bisector of q_i and q_r must go through the point y_1 as $y_1 - q_r = y_2 - q_i$. On the other hand, if q_i is inside the circle $(y_1, |y_1 - q_r|)$, then the bisector of q_r and q_i can meet at a point on the arc $\widehat{y_1y_2}$. In this case, the distance $|q_i - b_{ri2}|$ would have been chosen as the radius of the minimum bounding circle if the circle centered at b_{ri2} with a radius $|q_i - b_{ri2}|$ contains all the query points. Since $|q_r - b_{rt2}|$ is the minimum enclosing circles in B, the query point q_i cannot be inside the circle $(y_1, |y_1 - q_r|)$. Similarly, we can show that the circle $(y_2, |y_2 - q_t|)$ centered at point y_2 with a radius $|y_2 - q_t|$ cannot include all query points.

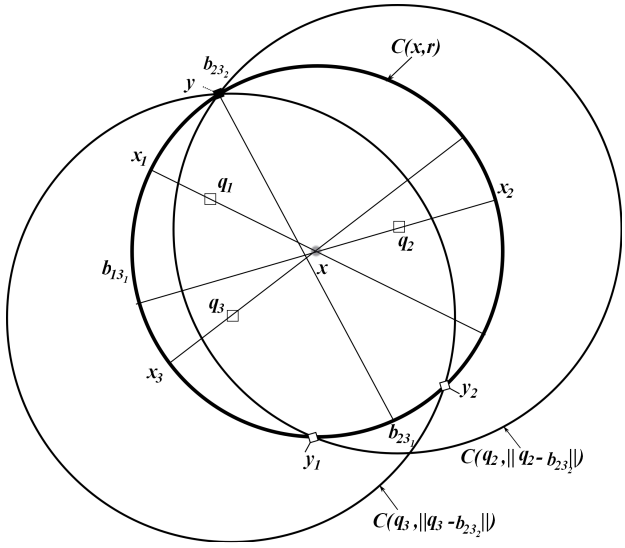

Fig. 18: The known circular region $C(x,r)$, query points $\{q_1, q_2, q_3\}$, and the BUAD $|q_2 - b_{232}|$.

We will now prove that the circle centered at any point $y' \in \widehat{y_1y_2}$ with a radius $|q_r - b_{rt2}|$ cannot contain all query points. Let us assume that the circle centered at a point y' on the arc $\widehat{y_1y_2}$ passes through a single query point q_i and contains all other query points. The distance $|y' - q_i|$ is greater than the minimum distance from q_i to ß. So there must be another query point q_j that is on the circle or inside the circle. In these cases, the bisector of q_i and q_j that meets at b_{ij_l} with ß would minimize the maximum distance for q_i and q_j. If the distance $|q_i - b_{ij_l}|$ is smaller than $|y' - q_i|$, the circle centered at b_{ij_l} with a radius $|q_i - b_{ij_l}|$ would have been selected as the smallest circle that encloses all the query points. Since $|q_r - b_{rt2}|$ is the minimum distance in B that encloses all query

points, the circle centered at any point $y' \in \widehat{y_1 y_2}$ with a radius equal to or less than $|q_r - b_{rt2}|$ cannot include all query points. Therefore, no point on arc $\widehat{y_1 y_2}$ can have a smaller aggregate distance than the distance $|q_r - b_{rt2}|$ for MAX.

Figure 18 shows an example that $|q_2 - b_{232}|$ is the BUAD as the circle centered at b_{232} with a radius $|q_2 - b_{232}|$ is the minimum bounding circle that encloses query points q_1, q_2, and q_3.

□

Thus, after computing the optimal BUAD for all subgroups of size m, the BUAD for the subgroup category m, $BUAD_m$ can be computed by taking the minimum of all $BUAD^i_m$s.

The BUAD for MIN: To find the BUAD for MIN, we need to find a point y from the boundary point set ß of $C(x,r)$ that minimizes the minimum distances from all query points, $\{q_1, q_2, \ldots, q_m\}$ to ß. That is, we need to find y that minimizes $\min_{i=1}^{m} |q_i - y|$. In this case, for a subgroup $sg_m = \{q_1, q_2, \ldots, q_m\}$, the proposed lower bound in Lemma 4.4 also gives the BUAD for function MIN as Lemma 4.4 finds the minimum of all minimum distances from query points to boundary point set ß of $C(x,r)$.

A.2. Cost Analysis

We compare our proposed methods with two competitive methods Combinatorial (COMB) and FANN in terms of number of subgroups that need to be computed. This measure can be considered a good proxy for the query CPU processing times.

The number of subgroups need to be computed by COMB is determined by the two parameters: (i) the number n of members in the group, and (ii) the minimum *relative* subgroup size r with respect to n, where $r \in [0,1]$. For a relative subgroup size r, the number of possible subgroups is given as

$$\frac{n!}{(\lfloor nr \rfloor)!(n - \lfloor nr \rfloor)!}.$$

Taking into account other subgroup sizes between $\lfloor nr \rfloor$ and n, we obtain the total number of possible subgroups as

$$\sum_{k=\lfloor nr \rfloor}^{n} \frac{n!}{k!(n-k)!}. \tag{1}$$

COMB iterates through all possible subgroup combinations and applies the GNN technique [Papadias et al. 2004] to each subgroup. In the worst case, for $r = 0$, we obtain the number of subgroups to be computed is 2^{n-1}.

The incremental methods INC-T-LB and INC-R-LB avoid consideration of all possible subgroups by incrementally expanding the search space and returning subgroups G in increasing order of the *gDist* from G to its nearest object. At each step of expansion, the INC-T-LB and INC-R-LB methods check the termination condition (Algorithm 1, Line 1.10). Specifically, INC-T-LB checks whether the tight BUAD is large enough and INC-R-LB checks whether the relaxed BUAD is large enough to guarantee the correctness of the current result.

Since the tight BUAD is guaranteed to be greater than or equal to the relaxed BUAD, the termination condition is easier to be satisfied using the tight BUAD than the relaxed one. However, for each subgroup size k, computing the tight BUAD requires iterating through $\binom{n}{k}$ subgroup combinations. When considering all subgroup sizes, we obtain $O(2^n)$ as the cost of each termination check.

The relaxed BUAD is computed by iterating through m query points (locations of group members) in the search space to find the aggregate distance from the query set $\{q_1, ..., q_m\}$ to the boundary of the search space. Since m is bounded by the total number n of query points, we can say that the worst case of checking the termination condition using the relaxed BUAD is $O(n)$. For the MQ method, the termination condition check is similar to that of INC-R-LB. However, since the termination condition needs to be checked for the expansion of the search space for each query point independently, the total cost of MQ is n times of the cost of INC-R-LB. Thus, the worst case

cost for MQ is $O(n^2)$. On the other hand, the FANN approach [Li et al. 2011] also uses a relaxed lower bound while computing the nearest neighbor for a fixed subgroup size $\lfloor nr \rfloor$. However, we need to repeat the FANN process for a range of values between $\lfloor nr \rfloor$ and n to obtain the results for the k-BEST-SUBGROUPS-NN query, and in the worst case, the value of r is 0. That is, we have to consider all possible subgroup sizes. Thus, the worst case cost for FANN is $O(n^2)$.

Note that the given complexity analysis only indicates how these algorithms scale as the dataset size n increases. Since big-o notation ignores constant factors associated with different types of operations, the complexity analysis is not indicative of which algorithm will be the fastest or the slowest one in a specific case. For example, the cost of computing the tight BAUD is exponential with respect to n where each operation involves only simple in-memory arithmetic calculations. While the cost for checking all possible combinations of sub-groups is also exponential for COMB, checking each combination involves executing an entire group NN query, which in turn incurs expensive index traversal operations.