The Flexible Group Spatial Keyword Query

Sabbir Ahmad[1], Rafi Kamal[1], Mohammed Eunus Ali[1], Jianzhong Qi[2], Peter Scheuermann[3], and Egemen Tanin[2]

[1] Department of CSE, Bangladesh Univ of Eng & Tech, Bangladesh
ahmadsabbir@cse.buet.ac.bd, rafikamalb@gmail.com,
eunus@cse.buet.ac.bd
[2] Department of CIS, University of Melbourne, Australia
jianzhong.qi@unimelb.edu.au, etanin@unimelb.edu.au
[3] Department of EECS, Northwestern University, USA
peters@eecs.northwestern.edu

Abstract. We propose the flexible group spatial keyword query and algorithms to process three variants of the query in the spatial textual domain: (i) the group nearest neighbor with keywords query, which finds the data object that optimizes the aggregate cost function for the whole group Q of size n query objects, (ii) the subgroup nearest neighbor with keywords query, which finds the optimal subgroup of query objects and the data object that optimizes the aggregate cost function for a given subgroup size m ($m \leq n$), and (iii) the multiple subgroup nearest neighbor with keywords query, which finds optimal subgroups and corresponding data objects for each of the subgroup sizes in the range $[m, n]$. We design query processing algorithms based on branch-and-bound and best-first paradigms. Finally, we provide theoretical bounds and conduct extensive experiments with two real datasets which verify the effectiveness and efficiency of the proposed algorithms.

1 Introduction

The *group nearest neighbor* (GNN) query [11] and its variants, the flexible aggregate nearest neighbor (FANN) [8] query and the consensus query [1] have been previously studied in spatial database domain. Given a set Q of n queries and a dataset D, a GNN query finds the data object that minimizes the aggregate distance (e.g., sum or max) for the group, whereas an FANN query finds the optimal subgroup of query points and the data object that minimizes the aggregate distance for a subgroup of size m, and a consensus query finds optimal subgroups and the data objects for each of the subgroup sizes in the range $[n', n]$. In all of these studies, the aggregate similarity is computed based on only spatial (or Euclidean) distances between a data point and a group of query points. In this paper, we address all the three variants of the above queries in the context of *spatial textual* domain, where both spatial proximity and keyword similarity for a *group or subgroups of users* to data points need to be considered. We call this class of query as the *flexible spatial keyword* query.

The flexible spatial keyword query has many applications in spatial and multimedia database domain. For example, in a *location-based social networks* (e.g.,

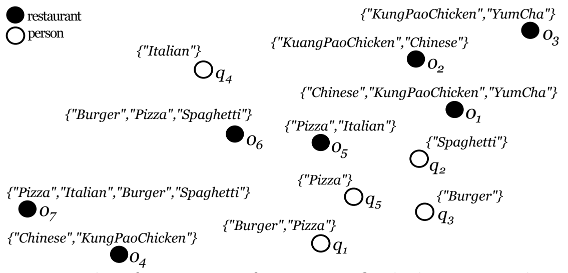

● restaurant
○ person
{"KungPaoChicken","YumCha"}
o_3
{"Italian"}
q_4
{"KuangPaoChicken","Chinese"}
o_2
{"Chinese","KungPaoChicken","YumCha"}
o_1
{"Burger","Pizza","Spaghetti"}
o_6
{"Pizza","Italian"}
o_5
{"Spaghetti"}
q_2
{"Pizza"}
q_5
{"Pizza","Italian","Burger","Spaghetti"}
o_7
{"Burger"}
q_3
{"Burger","Pizza"}
q_1
{"Chinese","KungPaoChicken"}
o_4

Fig. 1: An example of a group of users to find the optimal restaurant.

Foursquare), a group of users residing at their homes or offices can share their locations as spatial coordinates and their preferences as sets of keywords, and in order to find a Point of Interest (POI), e.g., restaurant or function venue, that optimizes a cost function composed of aggregate spatial distances and keyword similarities for the group. Since finding a POI that suits all group members might be difficult due to the diverse nature of choices, the group might prefer a result that is not optimal for the entire group, but is optimal for subset of it. In such cases, we need to find optimal a *subgroup* of users and a POI that minimizes the cost function for the subgroup.

Figure 1 shows an example, where a group of five friends $\{q_1, q_2, q_3, q_4, q_5\}$ is trying to decide a restaurant for a Sunday brunch. Each person provides his location and preferred type of food, represented by a set of keywords such as {"Burger", "Pizza"} or {"Italian"}, etc. There is a set of restaurants $\{o_1, o_2, ..., o_7\}$ to be selected from, and each restaurant is identified by its location and by a set of keywords describing the type of cuisine it offer, e.g., {"Pizza", "Italian"}. In general, it is preferred to find an answer that optimizes both spatial distance and keyword set dissimilarity at the same time, and o_7 is returned as the answer if we consider the whole group. However, if we allow leaving out a user, say q_4, then more answer candidates will become available. In particular, o_6 will now become the best choice of the subgroup $\{q_1, q_2, q_3, q_5\}$, as it covers all the keywords, and is the closest to members of the subgroup. In fact, leaving any other query user out (e.g., q_2) would not obtain a better cost function value. Therefore, $\{q_1, q_2, q_3, q_5\}$ is the optimal subgroup of size 4 and o_6 is the corresponding optimal data point.

We observe that in many practical applications relaxing the requirement, i.e., not including all the query objects, has potential benefits in finding good quality answer. For example, a company may want to find a location for holding a marketing campaign, where it is often desired that the selected place optimizes for at least 60% of the customers as it may be difficult to find a place that suits all customers. Similarly, in a multimedia domain, one may want to find an image that matches with a subgroup of query images, where an object or query image is represented as a point (in a high-dimensional space) and a set of tag-words. Generally, one may prefer the subgroup size to be maximized, and hence, it might be beneficial to explore the optimal solutions for different subgroup sizes.

The key challenge in processing the group spatial keyword queries is how to utilize both the spatial and keyword preferences and to efficiently prune the search space. Another major challenge is how to find the optimal subgroups of various sizes in one pass over the data set. Our contributions are as follows:

- We propose a new class of group queries in the spatial textual domain: (i) the group nearest neighbor with keywords (GNNK) query that finds the best data with respect to our cost function for the whole group, (ii) the subgroup nearest neighbor with keywords (SGNNK) that finds the optimal subgroup and the corresponding best POI for a given subgroup size of size m (with $m \leq n$, the group size)and (iii) the multiple subgroup nearest neighbor with keywords (MSGNNK) that returns in one pass the optimal subgroups and corresponding POIs for all subgroups of size m, where $n' \leq m \leq n$ and n' being the minimum subgroup size.
- We propose pruning strategies based on branch and bound as well as best-first strategies for these three queries. The resultant algorithms can process the queries in a single pass over the dataset.
- We provide theoretical bounds for our algorithms, and evaluate them through an extensive experimental evaluation on real datasets. The results demonstrate the effectiveness and efficiency of the proposed algorithms.

2 Problem Statement

Let D be a geo-textual dataset. Each object $o \in D$ is defined as a pair $(o.\lambda, o.\psi)$, where $o.\lambda$ is a location point and $o.\psi$ is a set of keywords. A query object q is similarly defined as a pair $(q.\lambda, q.\psi)$. Let $dist(q.\lambda, o.\lambda)$ be the spatial distance between q and o, and $similarity\_key(q.\psi, o.\psi)$ be the similarity between their keyword sets. We normalize both $dist(q.\lambda, o.\lambda)$ and $similarity\_key(q.\psi, o.\psi)$ so that their value lie between 0 and 1 (inclusive). The cost of o with respect to q is expressed in terms of their spatial distance and keyword set distance:

$$cost(q, o) = \alpha \cdot dist(q.\lambda, o.\lambda) + (1 - \alpha) \cdot (1 - similarity\_key(q.\psi, o.\psi))$$

Here, α is a user-defined parameter to control the preference of spatial proximity over keyword set similarity. The spatial distance is normalized by the maximum spatial distance between any pair of objects in the dataset, d_{max}. Thus,

$$dist(q.\lambda, o.\lambda) = euclidean\_distance(q.\lambda, o.\lambda)/d_{max}$$

Each keyword in the dataset is associated with a weight. The weight of each keyword is normalized by the maximum keyword weight w_{max} present in the dataset. Let $y.w$ be the weight of keyword y. Then the text relevance between q and o is the normalized sum of the weights of the keywords shared by q and o:

$$similarity\_key(q.\psi, o.\psi) = \frac{1}{|q.\psi|} \sum_{y \in q.\psi \cap o.\psi} \frac{y.w}{w_{max}}$$

We formulate the GNNK, SGNNK and MSGNNK queries as follows.

Definition 1. (*GNNK*). *Given a set D of spatio-textual objects, a set Q of query objects $\{q_1, q_2, ..., q_n\}$, and an aggregate function f, the GNNK query finds an object $o_i \in D$ such that for any $o' \in D \setminus \{o_i\}$,*

$$f(cost(q_j, o_i) : q_j \in Q) \leq f(cost(q_j, o') : q_j \in Q)$$

Definition 2. ($SGNNK$). *Given a set D of spatio-textual objects, a set Q of query objects $\{q_1, q_2, ..., q_n\}$, an aggregate function f, a subgroup size m ($m \leq n$), and the set SG_m of all possible subgroups of size m, the SGNNK query finds a subgroup $sg_m \in SG_m$ and an object $o_i \in D$ such that for any $o' \in D \setminus \{o_i\}$,*

$$f(cost(q_j, o_i) : q_j \in sg_m) \leq f(cost(q_j, o') : q_j \in sg_m)$$

and for any subgroup $sg'_m \in SG_m \setminus \{sg_m\}$,

$$f(cost(q_j, o_i) : q_j \in sg_m) \leq f(cost(q', o') : q' \in sg'_m)$$

Definition 3. ($MSGNNK$). *Given a set D of spatio-textual objects, a set Q of query objects $\{q_1, q_2, ..., q_n\}$, an aggregate function f, and minimum subgroup size n' ($n' \leq n$), the MSGNNK query returns a set S of $(n - n' + 1)$ $\langle subgroup, data\ object\rangle$ pairs such that, each pair $\langle sg_m, o_m\rangle$ is the result of the $SGNNK$ query with subgroup size m ($n' \leq m \leq n$).*

If the users are interested in the k-best POIs then the queries can be generalized as k-*GNNK*, k-*SGNNK* and k-*MSGNNK* queries. In this paper, we focus providing efficient solutions for the above queries for aggregate functions SUM ($\sum_{q_j \in Q} cost(q_j, o)$) and MAX ($\max_{q_j \in Q} cost(q_j, o)$).

3 Related Work

Group Nearest Neighbor Queries. The depth-first (DF) [13] and the best-first (BF) [7] algorithms are commonly used to process the k nearest neighbor (kNN) queries in spatial database. They assume the data objects to be indexed in a tree structure, e.g., the R-tree [6].

The group nearest neighbor (GNN) query [10] finds a data point that minimizes the aggregate distance for a group of query locations. SUM, MAX and MIN are commonly used aggregate functions. The generalization of the GNN query is the kGNN query, where k best group nearest neighbors are to be found. Several methods for processing GNN queries have been presented in [11].

The flexible aggregate nearest neighbor (FANN) query [8] is a generalization of the GNN query. It returns the data object that minimizes the aggregate distance to any subset of ϕn query points, where n is the size of the query group and $0 < \phi \leq 1$. The query also returns the corresponding subset of query points.

A query similar to the FANN query called the *consensus query* [1] is the main motivation of our paper. Given a minimum subgroup size m and a set of n query points, the consensus query finds objects that minimize the aggregate distance for all subgroups with sizes in the range $[m, n]$. A BF algorithm was proposed to process the consensus query.

Spatial Keyword Queries. The spatial keyword query consists of a query location and a set of query keywords. A spatio-textual data object is returned based on its spatial proximity to the query location and textual similarity with the query keywords. A number of indexing structures for processing the spatial keyword query have been proposed [3, 5, 9, 12, 15, 16]. Among them, the IR-tree [5, 9] has been shown to be a highly efficient one. The IR-tree augments each node of the R-tree with an inverted file corresponding to the keyword sets of the child nodes.

A variant of the spatial keyword query, called *spatial group keyword query* has been introduced [2,4]. It finds a group of objects that cover the keywords of a *single query* such that both the aggregate distance of the objects from the query location and the inter-object distances within the group are also minimized. Exact and approximate algorithms for three types of aggregate functions (SUM, MAX and MIN) have been presented in [2].

A work parallel to ours, the group top-k spatial keyword query (GLkT) has been proposed recently [14]. This paper presents a branch-and-bound technique to retrieve the top-k spatial keyword objects for only one group of queries. However, as we show in our experimental evaluation (Section 6), our best-first technique always outperforms the branch-and-bound method substantially even for a single group query.

4 Our Approach

This section presents our algorithms to process the GNNK, SGNNK and MSGNNK queries. The key challenge is to utilize the spatial distance and keyword preference together to constrain the search space as much as possible, since the performance of the algorithms is directly proportional to the search space (in both running time and I/O). Another challenge in the SGNNK and MSGNNK queries is to find the optimal subgroup from all possible subgroups.

4.1 Preliminaries

We use the IR-tree [5] to index our geo-textual dataset D. Other extensions of the IR-tree, such as the CIR-tree, the DIR-tree or the CDIR-tree [5] can be used as well. The IR-tree is essentially an inverted file augmented R-tree [6]. The leaf nodes of the IR-tree contain references to the objects from dataset D. Each leaf node has also a pointer to an inverted file index corresponding to the keyword sets of the objects stored in that node. The inverted file index stores a mapping from the keywords to the objects where the keywords appear. Each node N of the IR-tree has the form $(N.\Lambda,\ N.\Psi)$, where $N.\Lambda$ is the minimum bounding rectangle (MBR) that bounds the child node entries, and $N.\Psi$ is the union of the keyword sets in the child node entries.

The cost of an IR-tree node is defined similarly to the cost of a data object:

$$cost(q, N) = \alpha\ min\_dist(q.\lambda, N.\Lambda) + (1-\alpha)\ (1 - similarity\_key(q.\psi, N.\Psi))$$

Here, $min\_dist(q.\lambda, N.\Lambda)$ is the minimum spatial distance between the query object location $q.\lambda$ and the MBR of N; $similarity\_key(q.\psi, N.\Psi)$ is the textual similarity between the query keywords and the keywords of the node. The cost of an IR-tree node gives a lower bound over the cost of its children, as formalized by the following lemma:

Lemma 1. *Let N be an IR-tree node and q be a query object. If N_c is a child of N, then $cost(q, N) \leq cost(q, N_c)$.*

Proof. The child N_c can either be a data object or an IR-tree node. In either case $min\_dist(q.\lambda, N.\Lambda)$ is smaller than or equal to that of N_c according to the R-tree structure. Meanwhile, the keyword set of N_c is a subset of the keyword set of N. Thus, N will have a higher (or equal) textual similarity value (and hence lower keyword set distance) with the query keywords. Overall, we have $cost(q, N) \leq cost(q, N_c)$. □

4.2 Branch and Bound Algorithms for GNNK and SGNNK

Traditional nearest neighbor algorithms access the data indexed in a spatial index (e.g., R-tree) and restricts its search space by pruning bounds [13]. We extend the idea to design two branch and bound algorithms for the GNNK and SGNNK queries. These algorithms work as the baseline in our experiments.

Branch and Bound Algorithm for GNNK. We use the following heuristic to prune the unnecessary nodes while searching the IR-tree for the best object with the minimum aggregate cost.

Heuristic 1. *A node N can be safely pruned if its aggregate cost with respect to the query set Q is greater than or equal to the smallest cost of any object retrieved so far.*

This heuristic is derived from Lemma 1. As f is a monotonic function and $cost(q, N) \leq cost(q, N_c)$ for any child N_c of N, $f(cost(Q, N)) \leq f(cost(Q, N_c))$. Let min_cost be the smallest cost of any data object retrieved so far. Then $f(cost(Q, N)) \geq min\_cost$ implies that the cost of any descendant of N is greater than or equal to min_cost, and we can safely prune N.

The branch and bound algorithm for GNNK is based on the heuristic and denoted by GNNK-BB. The algorithm keeps a stack and inserts the child nodes of the IR-tree into the stack, if the aggregate cost of the node is less than min_cost. After all the nodes are explored, the leaf node for the min_cost is returned.

Branch and Bound Algorithm for SGNNK. We design a similar branch and bound algorithm named SGNNK-BB for the SGNNK query. The following heuristic is used for pruning.

Heuristic 2. *Let N be an IR-tree node and m be the required subgroup size. If sg_m is the best subgroup of size m, and min_cost is the smallest cost of any size-m subgroup retrieved so far, we can safely prune N if $f(cost(sg_m, N)) \geq min\_cost$.*

This heuristic is derived from Lemma 1. Let N_c be a child of N and sg'_m be the best subgroup corresponding to N_c. Then we have $f(cost(sg'_m, N)) \leq f(cost(sg'_m, N_c))$. Meanwhile sg_m is the best subgroup for N among all possible subgroups of size m. Thus, $f(cost(sg_m, N)) \leq f(cost(sg'_m, N))$.

The above two inequalities imply that $f(cost(sg_m, N)) \leq f(cost(sg'_m, N_c))$, i.e., the aggregate cost for the best size-m subgroup of N is lower than or equal to that of the best size-m subgroup of any of its children. Therefore, if $f(cost(sg_m, N)) \geq min\_cost$, $f(cost(sg_m, N_c))$ will also be greater than or equal to min_cost, and we should prune N. The overall tree traversal procedure is similar to that of the GNNK-BB algorithm. The difference is in the calculation of the optimization function, where the optimization function value is computed based on the the top-m queries with the lowest costs.

4.3 Best-first Algorithms for GNNK and SGNNK

Branch and bound techniques may access unnecessary nodes during query processing. To improve the query efficiency by reducing disk accesses, we propose in this section best-first search techniques that only access the necessary nodes.

Best-first algorithm for GNNK. The best-first procedure for the GNNK query is denoted by GNNK-BF. This algorithm uses a minimum priority queue to maintain the nodes/objects to be visited according to their aggregate costs. If an intermediate node (leaf node) is popped, all the child nodes (child objects) are pushed into the queue. When an object is first popped from the queue, it denotes the minimum cost object and is returned as the query result. The algorithm is not shown due to space limitation.

Algorithm 1 SGNNK-BF (R, Q, m, f)

```
INPUT: IR-tree index $R$, $n$ query points $Q = \{q_1, q_2, ..., q_n\}$, subgroup size $m$, $f$.
OUTPUT: A data object $o$ and a set of $m$ query points $sg_m$ that minimize $f(cost(sg_m, o))$
1: Initialize a new min priority queue $P$ and $P.push(root, 0)$
2: repeat
3:     $E \leftarrow P.pop()$
4:     if $E$ is an intermediate node $N$ then
5:         for all $N_c$ in $N.children$ do
6:             Compute $cost(q_1, N_c), ..., cost(q_n, N_c)$
7:             $sg_m \leftarrow$ first $m$ query points with the lowest cost values
8:             $P.push(N_c, f(cost(sg_m, N_c)))$
9:     else if $E$ is a leaf node $N$ then
10:        for all $o$ in $N.children$ do
11:            Compute $cost(q_1, o), ..., cost(q_n, o)$
12:            $sg_m \leftarrow$ first $m$ query points with the lowest cost values
13:            $o.best\_subgroup = sg_m$
14:            $P.push(o, f(cost(sg_m, o))$
15:    else if $E$ is a data object $o$ then
16:        return $(o, o.best\_subgroup)$
17: until $P$ is empty
18: return $null$
```

Best-first Algorithm for SGNNK. The best-first algorithm for the SGNNK query, denoted by SGNNK-BF, is similar to GNNK-BF algorithm. Here, the optimization function is computed for top-m queries. Best subgroup is chosen from the lowest m query points, and pushed into the priority queue. For an intermediate node, aggregate costs and best subgroup are calculated for all the child nodes of the node. For a leaf node, it is done for all the children objects, and then pushed into the priority queue. When an object is first popped, it is returned as the result. The pseudocode is shown in Algorithm 1.

4.4 Algorithms for MSGNNK

To process the MSGNNK query with a minimum subgroup size m, we can run SGNNK-BF $n - m + 1$ times (for subgroup sizes $m, m + 1, ..., n$) and return the combined results. We call this the MSGNNK-N algorithm. However, MSGNNK-N requires accessing the dataset $n-m+1$ times, which is too expensive. To avoid this repeated data access, we design an algorithm based on best-first method that

Algorithm 2 MSGNNK-BF (R, Q, m, f)

```
INPUT: Index $R$ of objects, query set $Q$, min subgroup size $m(m \leq n)$, and $f$.
OUTPUT: A set of $\langle data\_object, subgroup \rangle$ pairs $\langle o_k^*, sg_k^* \rangle$ for all subgroup sizes between $m$ and
    $n$ (inclusive), where $\langle o_k^*, sg_k^* \rangle$ minimizes $f(cost(sg_k, o))$.
 1: Initialize a new min priority queue $P$ and $P.push(root, 0)$
 2: $min\_costs[i] \leftarrow \infty$ and $root.query\_costs[i] \leftarrow 0$ for $m \leq i \leq n$
 3: repeat
 4:     $E \leftarrow P.pop()$
 5:     if $\exists i \in [m, n]$: $E.query\_costs[i] < min\_costs[i]$ then
 6:         if $E$ is an intermediate node then
 7:             for all $N_c$ in $E.children$ do
 8:                 Compute $cost(q_1, N_c), ..., cost(q_n, N_c)$
 9:                 $total\_cost \leftarrow 0$
10:                 for $i = m \rightarrow n$ do
11:                     $sg_i \leftarrow$ top $i$ lowest cost query points
12:                     $total\_cost$ += $f(cost(sg_i, N_c))$
13:                     $N_c.query\_costs[i] = f(cost(sg_i, N_c))$
14:                 if $f(cost(sg_i, N_c)) < min\_costs[i]$ for any subgroup size $i \in [m, n]$ then
15:                     $P.push(N_c, total\_cost)$
16:         else if $E$ is a leaf node then
17:             for all $o$ in $N.children$ do
18:                 Compute $cost(q_1, o), ..., cost(q_n, o)$
19:                 for $i = m \rightarrow n$ do
20:                     $sg_i \leftarrow$ top $i$ lowest cost query points
21:                     if $f(cost(sg_i, o)) < min\_costs[i]$ then
22:                         $min\_costs[i] \leftarrow f(cost(sg_i, o))$
23:                         $best\_objects[i] \leftarrow o$
24:                         $best\_subgroups[i] \leftarrow sg_i$
25: until $P$ is empty
26: return $best\_objects, best\_subgroups$
```

can find the best data objects for all subgroup sizes between m and n in a single pass over the dataset. Algorithm 2 summarizes the proposed procedure, denoted as MSGNNK-BF. The algorithm is based on the following heuristic.

Heuristic 3. *Let N be an IR-tree node and m be the minimum subgroup size. Let sg_i be the best subgroup of size i ($m \leq i \leq n$), and min_cost_i be the smallest cost for subgroup size i from any object retrieved so far. We can safely prune N if $f(cost(sg_i, N)) \geq min\_cost_i$ for any i.*

The proof of correctness is straightforward based on Heuristic 1 and Heuristic 2, and is omitted due to space limit.

A relaxed pruning bound. A possible simplification of Heuristic 3 is to only test whether $f(cost(sg_m, N)) \geq min\_cost_n$, i.e., whether the best subgroup of size m corresponding to N has a cost lower than the min_cost for the whole group of size n found so far. If this holds, then N can be safely pruned, as formalized by the following heuristic.

Heuristic 4. *Let N be an IR-tree node and m be the minimum subgroup size. Let sg_m be the best subgroup of size m corresponding to N, and min_cost_n be the smallest cost for the whole group of size n from any object retrieved so far. We can safely prune N if $f(cost(sg_m, N)) \geq min\_cost_n$.*

The proof is straightforward and thus omitted. Note that, while this heuristic simplifies the node pruning computation (Lines 12 to 17 in Algorithm 2), it also relaxes the pruning bound, which may cause more nodes to be processed.

5 Cost Analysis

We analytically compare the I/O cost and CPU cost of the proposed algorithms. We use the following notations in the analysis. Let C_m be the maximum number of entries in a disk block, C_e be the effective capacity of the IR-tree used to index the dataset D, and $|D|$ be the size of D. We assume that an IR-tree node size equals a disk block. I/O cost and CPU cost of the preloading is denoted by io_i and cpu_i, respectively. We quantify the percentage of pruned nodes in the tree traversal as the pruning power, denoted by w and is represented by w_{gb}, w_{gf}, w_{sb}, w_{sf}, and w_{mb} for GNNK-BB, GNNK-BF, SGNNK-BB, SGNNK-BF, and MSGNNK-BF, respectively. We denote the I/O cost of accessing the inverted index by io_l, and the associated CPU cost by cpu_l. Per node CPU cost of GNNK-BB and GNNK-BF is denoted by cpu_g, SGNNK-BB and SGNNK-BF by cpu_s and MSGNNK-BF by cpu_m. Table 1 summarizes the analytical results. Details calculation of cost analysis is omitted due to page limitation.

Table 1: Summary of Costs

Algorithm	I/O	CPU
GNNK-BB	$io_i + (1 - w_{gb})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (1 - w_{gb})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_g + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$
GNNK-BF	$io_i + (1 - w_{gf})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (1 - w_{gf})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_g + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$
SGNNK-BB	$io_i + (1 - w_{sb})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (1 - w_{sb})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_s + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$
SGNNK-BF	$io_i + (1 - w_{sf})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (1 - w_{sf})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_s + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$
MSGNNK-N	$io_i + (n - m + 1)(1 - w_{sf})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (n - m + 1)(1 - w_{sf})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_s + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$
MSGNNK-BF	$io_i + (1 - w_{mb})(\frac{\lvert D\rvert}{C_e - 1} + \frac{\lvert D\rvert}{C_e} \cdot io_l)$	$cpu_i + (1 - w_{mb})(\frac{\lvert D\rvert}{C_e - 1} \cdot cpu_m + \frac{\lvert D\rvert}{C_e} \cdot cpu_l)$

6 Experimental Evaluation

6.1 Experimental Settings

We evaluate the performance of the proposed algorithms. The BB algorithms are used as the baseline to compare with the BF algorithms for the GNNK and SGNNK queries. We use the MSGNNK-N algorithm as the baseline algorithm for the MSGNNK queries, and compare it with the MSGNNK-BF algorithm.

Table 2: Dataset properties

Parameter	**Flickr**	**Yelp**
Dataset size	1,500,000	60,667
Number of unique keywords	566,432	783
Total number of keywords	11,579,622	176,697
Avg. number of keywords per object	7.72	2.91

Table 3: Query parameters

Parameter name	Values	Default Value
Number of queried data points (k)	1, 10, 20, 30, 40, 50	10
Query group size (n)	10, 20, 40, 60, 80	10
Subgroup size (m, $\%n$)	40%, 50%, 60%, 70%, 80%	60%
Number of query keywords	1, 2, 4, 6, 8, 10	4
Size of the query space	.001%, .01%, .02%, .03%, .04%, .05%	0.01%
Size of the query keyword set	1%, 2%, 3%, 4%, 5%	3%
Spatial vs. textual preference (α)	0.1, 0.3, 0.5, 0.7, 1.0	0.5
Dataset Size (Flickr)	1M, 1.5M, 2M, 2.5M	1.5M

We use two real datasets from Yahoo! Flickr[4] and Yelp[5] in our experiments. The properties of these two datasets are detailed in Table 2. We generate 20 groups of query objects for each experiment and average the results. Each query object contains a location and a set of keywords. Locations are generated uniformly inside a square query space area. For generating the query keywords, a subset of keywords from all keywords inside the query space is first chosen, and then the required of number of keywords are selected from this subset. The parameters that are varied are shown in Table 3.

We use the IR-tree to index the datasets, which is disk resident. The fanout of the IR-tree is chosen to be 50, and the page size is 4KB. All the algorithms are implemented in Java and the experiments are conducted on a Core i7-4790 CPU @ 3.60 GHz with 4 GB of RAM. We measure the running time and the I/O cost (number of disk page accesses), where the running time includes the computation and I/O time. We use Flickr as our default dataset.

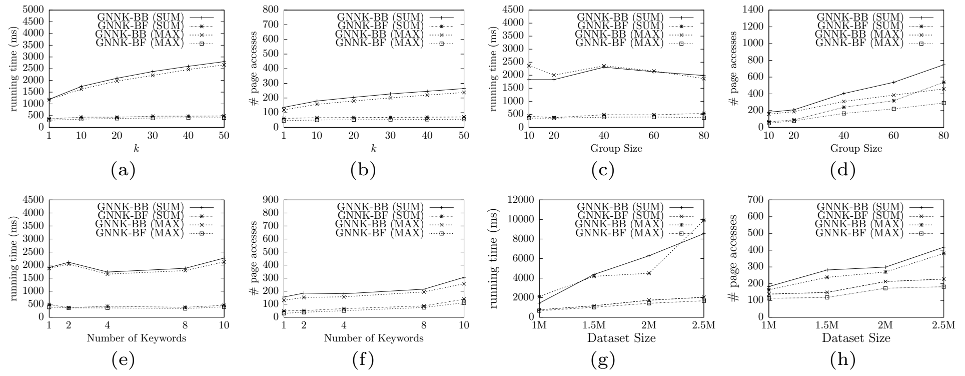


Fig. 2: The effect of varying k (a-b), query group size (c-d), number of query keywords (e-f), query keyword set size (g-h) and dataset size (i-j) in running time and I/O

6.2 The GNNK Query Algorithms

We conduct seven sets of experiments to evaluate the performance of GNNK-BB and GNNK-BF. In each set of experiments, one parameter is varied while all other parameters are set to their default values. GNNK-BF outperforms GNNK-BB in all experiments both in terms of running time and I/O cost.

Varying k. Figure 2 (a-b) shows that for both GNNNK-BB and GNNK-BF, the processing time and the I/O cost increase with the increase of k. For both SUM and MAX, on average GNNK-BF runs 3.5 times faster than GNNK-BB. The I/O cost of GNNK-BF is much less than that of GNNK-BB as GNNK-BF only accesses the necessary nodes.

Varying Query Group Size. Figure 2 (c-d) shows the effect of the query group size (n). The query processing costs of both algorithms increase as the value of n increases. On average, GNNK-BF runs approximately 4 times faster than GNNK-BB.

[4] `https://webscope.sandbox.yahoo.com`

[5] `https://www.yelp.com/academic_dataset`

Varying Number of Query Keywords. Figure 2 (e-f) shows the effect of the number of keywords in each query object. GNNK-BF again outruns GNNK-BB in all the experiments. Also, the query processing costs of both algorithms increase as the number of keywords in each query object increases. This can be explained by that a larger set of query keywords takes more time to compute the aggregate cost function.

Varying Query Space Size. We observe that the running time of our algorithms remains almost constant with the change of the query space area (not shown in graphs). Since varied query space areas are insignificant in compared to the data space, we do not observe any significant change in this experiment.

Varying Query Keyword Set Size. We see that the running time of our algorithms do not follow any regular pattern with the change of the query keyword set size and remains relatively stable for varying of query keyword set size (the subset of keywords from where the query keywords are generated).

Varying α. We observe that, as α increases, the query costs decrease. A larger α means that spatial proximity is deemed more important than textual similarity. When α increases, the impact of the keyword similarity becomes smaller and algorithms converge faster (not shown in graphs).

Varying Dataset Size. Figure 2 (g-h) shows the effect of varying number of objects. Both running time and I/O cost of our proposed algorithms increase at a lower rate than the baseline algorithms.

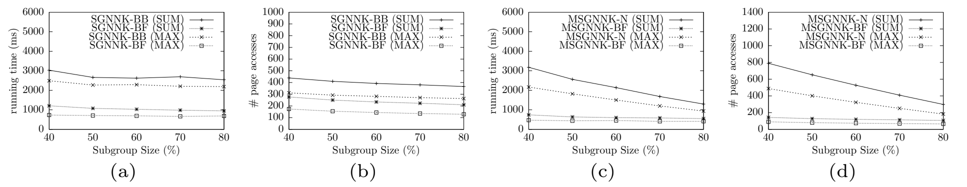


Fig. 3: The effect of varying subgroup size m (a-b) and minimum subgroup size (c-d)

6.3 The SGNNK Query Algorithms

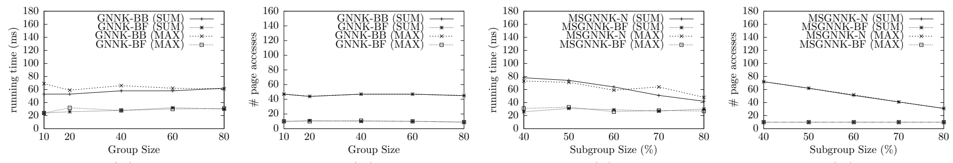
We performed experiments on SGNNK-BB and SGNNK-BF, by varying query group size, subgroup size, number of query keywords, query space size, query keyword set size, k, dataset size, and α. SGNNK-BF outperforms SGNNK-BB in all the experiments. For space constraints, we only show the effect of varying the subgroup size (in % n) in Figure 3 (a-b). On average, SGNNK-BF runs 3.5 times faster and takes 40% less I/O than SGNNK-BB.

6.4 The MSGNNK Query Algorithms

We performed similar experiments on MSGNNK-N and MSGNNK-BF. In all the experiments MSGNNK-BF significantly outperforms MSGNNK-N. Due to space constraints, we only show the effect of varying the minimum subgroup size (in percentage of n) in Figure 3 (c-d). When the minimum subgroup size increases, the running time of both algorithms decrease as expected. Meanwhile, the costs of MSGNNK-BF change in a much smaller scale, which demonstrates the better scalability of MSGNNK-BF. On average, MSGNNK-BF runs about 4 times faster than MSGNNK-N.

6.5 Experiments on Yelp dataset

We have run the same set of experiments as mentioned above on the Yelp dataset. All of our experimental results show similar trends in both datasets. Due to page limitations, we only present the experimental results for varying group size for GNNK queries and minimum subgroup size for MSGNNK queries with Yelp dataset in Figure 4 (a-b) and Figure 4 (c-d), respectively.



(a) (b) (c) (d)

Fig. 4: The effect of varying group size (a-b) and minimum subgroup size (c-d)

7 Conclusion

In this paper, we have presented a new type of group spatial keyword query suitable for a collaborative environment. This query aims to find the best POI that minimizes the aggregate distance and maximizes the text relevancy for a group of users. We have studied three instances of this query, which return (i) the best POI for the whole group, (ii) the optimal subgroup with the best POI given a subgroup size m, and (iii) the optimal subgroups and the corresponding best POIs of different subgroup sizes in $m, m+1, ..., n$. In all these queries, our proposed best-first approach runs approximately 4 times faster (on average) than the branch and bound approach for both real datasets.

References

1. M. E. Ali, E. Tanin, P. Scheuermann, S. Nutanong, and L. Kulik. Spatial consensus queries in a collaborative environment. *TSAS*, 2(1):3:1–3:37, 2016.
2. X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *TODS*, 40(2):13, 2015.
3. X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1-2):373–384, 2010.
4. X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.
5. G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
6. A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.
7. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.
8. Y. Li, F. Li, K. Yi, B. Yao, and M. Wang. Flexible aggregate similarity search. In *SIGMOD*, pages 1009–1020, 2011.
9. Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.
10. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

11. D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2):529–576, 2005.
12. J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222. 2011.
13. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *ACM SIGMOD Record*, volume 24, pages 71–79, 1995.
14. K. Yao, J. Li, G. Li, and C. Luo. Efficient group top-k spatial keyword query processing. In *ApWeb*, pages 153–165, 2016.
15. D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.
16. Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung. INSPIRE: A framework for incremental spatial prefix query relaxation. *TKDE*, 27(7):1949–1963, 2015.