Efficient Processing of Spatial Group Keyword Queries

XIN CAO, Queen's University Belfast
GAO CONG and TAO GUO, Nanyang Technological University
CHRISTIAN S. JENSEN, Aalborg University
BENG CHIN OOI, National University of Singapore

With the proliferation of geo-positioning and geo-tagging techniques, spatio-textual objects that possess both a geographical location and a textual description are gaining in prevalence, and spatial keyword queries that exploit both location and textual description are gaining in prominence. However, the queries studied so far generally focus on finding individual objects that each satisfy a query rather than finding groups of objects where the objects in a group together satisfy a query.

We define the problem of retrieving a group of spatio-textual objects such that the group's keywords cover the query's keywords and such that the objects are nearest to the query location and have the smallest inter-object distances. Specifically, we study three instantiations of this problem, all of which are NP-hard. We devise exact solutions as well as approximate solutions with provable approximation bounds to the problems. In addition, we solve the problems of retrieving top-k groups of three instantiations, and study a weighted version of the problem that incorporates object weights. We present empirical studies that offer insight into the efficiency of the solutions, as well as the accuracy of the approximate solutions.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Spatial keyword query, spatial group keyword query

ACM Reference Format:
Xin Cao, Gao Cong, Tao Guo, Christian S. Jensen, and Beng Chin Ooi. 2015. Efficient processing of spatial group keyword queries. ACM Trans. Datab. Syst. 40, 2, Article 13 (June 2015), 48 pages.
DOI: http://dx.doi.org/10.1145/2772600

1. INTRODUCTION

With the proliferation of geo-positioning techniques, such as GPS or techniques that exploit the wireless communication infrastructure, accurate user location is increasingly available. Similarly, increasing numbers of objects are available on the Web that have an associated geographical location and textual description. Such *spatio-textual objects* include Web content that represents stores, tourist attractions, restaurants, hotels, and businesses.

This development gives prominence to *spatial keyword queries* [Cao et al. 2012b; Chen et al. 2006, 2013; Cong et al. 2009; De Felipe et al. 2008]. A typical such query

This work was supported in part by a Singapore MOE AcRF Tier 2 Grant (ARC30/12), by a grant from the Obel Family Foundation, and by the Geocrowd Initial Training Network, funded by the European Commission as an FP7-PEOPLE Marie Curie Action under grant agreement no. 264994.
Authors' addresses: X. Cao, School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast; G. Cong (corresponding author) and T. Guo, School of Computing Engineering, Nanyang Technological University, Singapore; email: gaocong@ntu.edu.sg; C. S. Jensen, Department of Computer Science, Aalborg University, Denmark; B. C. Ooi, Department of Computer Science and School of Computing, National University of Singapore, Singapore.

DOI: http://dx.doi.org/10.1145/2772600

takes a location and a set of keywords as arguments and returns the single spatio-textual object that best matches these arguments.

We observe that user needs may exist that are not easily satisfied by a single object, but where groups of objects may combine to meet the user needs. Put differently, the objects in a group collectively meet the user needs. For instance, consider a tourist who is on vacation in an unfamiliar city. She wishes to have dinner at a Japanese restaurant, do some shopping, and have a drink in a bar near her hotel. For convenience, she prefers locations within walking distance of each other. Traditional spatial keyword queries return a single object which does not meet the tourist's needs. Rather, a query that returns a group of objects that together meet her needs is called for. As another example, an organizer wishes to assemble a team of helpers for a particular volunteer task. The helpers must together possess the capabilities required for successful completion of the task. This requires that the helpers should be close to the location of the organizer and close to each other as well.

To address the need for collective answers to spatial keyword queries as indicated in the examples, we assume a database of spatio-textual objects and then consider the problem of how to retrieve a group of spatio-textual objects that collectively meet the user's needs, given as a location and a set of keywords: (1) the textual description of the group of objects cover the query keywords; (2) the objects are close to the query point; and (3) the objects in the group are close to each other.

Specifically, given a set of spatio-textual objects D and a query $q = (\lambda, \psi)$ where λ is a location and ψ is a set of keywords, we consider three instantiations of the *spatial group keyword* query. It turns out that the subproblems corresponding to the three instantiations are all NP-hard.

(1) We aim to find a group of objects χ that cover the keywords in q such that the sum of their spatial distances to the query is minimized.
(2) We aim to find a group of objects χ that cover the keywords in q such that the sum of the maximum distance between an object in χ and q and the maximum distance between two objects in χ is minimized.
(3) We aim to find a group of objects χ that cover the keywords in q such that the sum of the minimum distance between an object in χ and q and the maximum distance between two objects in χ is minimized.

The first subproblem can be reduced from the weighted set cover problem. We propose a greedy algorithm that provides an approximate solution to the problem. This algorithm utilizes a spatial-keyword index such as the IR-tree [Cong et al. 2009] to prune the search space, and has a provable approximation bound. In some cases, the number of keywords in a query q may not be large. For such cases, we also propose an exact algorithm that explores the search space of the keywords, rather than enumerating the combinations of objects in the database. In particular, it uses dynamic programming to generate the optimal group. We also utilize a spatial-keyword index to further improve the performance of the exact algorithm by exploiting a series of pruning strategies.

The second subproblem can be reduced from the 3-SAT problem. We develop two approximation algorithms based on a spatial-keyword index with provable approximation bounds. The first approximation algorithm has an approximation ratio of 3, while the second an approximation ratio of 1.8. We also develop an exact algorithm that exploits a spatial-keyword index to prune the search space. As shown in the experimental study, our exact algorithm consistently outperforms the recently proposed best-known algorithm for the same problem [Long et al. 2013].

The third subproblem can be reduced from the 3-SAT problem as well. We develop an approximation algorithm that has an approximation ratio of 3 and an exact algorithm based on a spatial-keyword index enhanced with several pruning strategies.

In order to provide users with more options, for all three subproblems, we propose to find the top-k groups covering all the query keywords and ranked according to their costs. When a user is not satisfied with the returned top-1 group of objects, returning more groups may better satisfy the user's needs. We extend the exact algorithms of the three subproblems to answer the top-k spatial group keyword queries.

Finally, we also study a weighted variant of the spatial group keyword query, where we assume that the objects have weights. In the original query, the cost function is computed only based on distances, and the objects are treated equally. The weighted spatial group keyword query also takes into account object weights when computing the cost of a group. The weight of an object can be a user-contributed rating (e.g., Zagat[1] has huge amounts of user-contributed restaurant ratings), or it can be the text relevance of the object to the query. For example, an object close to the query location but with a low rating may be less preferred by users than a further-away object that is highly rated. The text relevance of an object to the query can be computed using any information retrieval model, such as the language model [Zhai and Lafferty 2004]. We study the weighted variant of all three instantiations of the spatial group keyword query.

The rest of the article is organized as follows. Section 2 formally defines the problem and establishes its computational complexities. Section 3 reviews the IR-tree indexing structure utilized in the article. Section 4 presents an approximate algorithm and several exact algorithms for the first subproblem, Section 5 presents two approximate algorithms and an exact algorithm for the second subproblem, and Section 6 presents an approximate algorithm and an exact algorithm for the third subproblem. Section 7 presents the algorithms for the top-k spatial group keyword query, and Section 8 introduces algorithms for the weighted version of the spatial group keyword query. We report on empirical studies in Section 9. Finally, we cover related work in Section 10 and offer conclusions and research directions in Section 11.

2. PROBLEM STATEMENT

Let S be a set of keywords. The keywords may capture user preferences or required project partner capabilities, depending on the application. Let D be a database consisting of m spatio-textual objects. Each object o in D is associated with a location $o.\lambda$ and a set of keywords $o.\psi$, $o.\psi \subset S$, that describes the object (e.g., the menu of a restaurant or the skills of a possible project partner).

Definition 2.1 (*Spatial Group Keyword Query*). A *spatial group keyword* (SGK) query q is of the form $\langle q.\lambda, q.\psi \rangle$, where $q.\lambda$ is a location and $q.\psi$ is a set of keywords. It finds a group of objects χ, $\chi \subseteq D$, such that $\cup_{r \in \chi} r.\psi \supseteq q.\psi$ and such that $\mathsf{Cost}(\chi)$ is minimized.

Definition 2.2 (*Feasible Group*). Given an SGK query q, if the union of the keywords of the objects in a group can cover all the keywords in $q.\psi$, we call such a group a *feasible group* or a *feasible solution*.

We proceed to present cost functions. Given a set of objects χ, a cost function has two weighted components, gives as

$$\mathsf{Cost}(q, \chi) = \alpha C_1(q, \chi) + (1 - \alpha) C_2(\chi), \tag{1}$$

[1]http://www.zagat.com/.

where $C_1(\cdot,\cdot)$ is dependent on the distance of the objects in χ to the query object and $C_2(\cdot)$ characterizes the inter-object distances among the objects in χ. This type of cost function is capable of expressing that the result object should be near the query location ($C_1(\cdot,\cdot)$), that the result objects should be near to each other ($C_2(\cdot)$), and that these two aspects are given different weights (α).

Many functions could be used for $C_1(\cdot,\cdot)$ and $C_2(\cdot)$. In order to characterize the distance between the group χ and the query q, we can compute the minimum, maximum, and total distance of all the objects in χ to q. For computing $C_2(\cdot)$, we consider the diameter of χ, that is, the maximum distance between each pair of objects in χ. Consequently, we obtain the following combinations as our cost functions.

(1) $\mathsf{Cost}(q,\chi) = \sum_{o\in\chi} \mathsf{Dist}(o,q)$;

(2) $\mathsf{Cost}(q,\chi) = \alpha \max_{o\in\chi}(\mathsf{Dist}(o,q)) + (1-\alpha)\max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1,o_2))$;

(3) $\mathsf{Cost}(q,\chi) = \alpha \min_{o\in\chi}(\mathsf{Dist}(o,q)) + (1-\alpha)\max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1,o_2))$;

(4) $\mathsf{Cost}(q,\chi) = \alpha \sum_{o\in\chi} \mathsf{Dist}(o,q) + (1-\alpha)\max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1,o_2))$;

(5) $\mathsf{Cost}(q,\chi) = \max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1,o_2))$.

Function (5) only considers the inter-distances between each pair of objects in the group, and it turns out to be the mCK query which has been studied already [Guo et al. 2015; Zhang et al. 2009, 2010]. We believe the following four instantiations of the cost function $\mathsf{Cost}(q,\chi)$, that correspond to cost functions (1)–(4), have meaningful applications.

Cost function (1). The cost function, called SUM, is the sum of the distance between each object in χ and the query location. SUM may fit with applications where the user needs to return to the query location in-between visiting each object. For example, the user wants to rest in the hotel in-between going to the gym and the theater.

Cost function (2). The first term in this cost function, called MAX+MAX, is the maximum distance between any object in χ and the query location q, and the second term is the maximum distance between two objects in χ (this can be understood as the diameter of the result). This cost function may be used when the users would like to visit result objects one by one, without returning to the query location in-between.

Cost function (3). The first term in this cost function, called MIN+MAX, is the minimum distance between any object in χ and the query location q, and the second term is the maximum distance between any two objects in χ. This function is preferable when users expect the nearest object in a result group to be close to the query location.

Cost function (4). The first term in this cost function, called SUM+MAX, is the total distance between all objects in χ and the query location q, and the second term is the maximum distance between any two objects in χ. This cost function is preferable in scenarios such as that of finding helpers for a task if the helpers will meet often at the organizer's location and will visit each other as well.

The following theorem demonstrates the hardness of answering the spatial group keyword query.

THEOREM 2.3. *The problem of answering the spatial group keyword query using any of the four types of cost functions is NP-hard.*

PROOF.

(1) We first consider the SUM cost function. We prove the theorem by a reduction from the weighted set cover problem. An instance of the weighted cover problem consists of a universe $U = \{1, 2, \ldots, n\}$ of n elements and a family of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$,

where $S_i \subseteq U$ and each S_i is associated with a positive cost C_{S_i}. The problem is to find a subset $\mathcal{F}$ of $\mathcal{S}$ such that $\cup_{S_i \in \mathcal{F}} S_i = U$ and such that its cost $\sum_{S_i \in \mathcal{F}}(C_{S_i})$ is minimized.

To reduce this problem to that of answering the SUM spatial group keyword, we observe that each element in U corresponds to a keyword in $q.\phi$, that each S_i corresponds to a spatial object o_i containing a set of keywords, and that the weight of S_i is $dist(q, o_i)$, that is, the distance between the query and object o_i. It is easy to show that there exists a solution to the weighted set cover problem if and only if there exists a solution to query q.

(2) Considering next the MAX+MAX cost function, we prove the theorem by a reduction from the 3-SAT problem. An instance of the 3-SAT problem consists of $\Phi = C_1 \wedge C_2, \ldots, \wedge C_l$, where each clause $C_j = x_j \vee y_j \vee z_j$, and $\{x_j, y_j, z_j\} \in \{e_1, \bar{e_1}, e_2, \bar{e_2}, \ldots, e_n, \bar{e_n}\}$. The decision problem is to determine whether we can assign a truth value to each of the literals (e_1 through e_n) such that Φ is *true*.

We reduce this problem to an instance of the problem of answering the MAX+MAX spatial group keyword query (with $\alpha = 0.5$), which is to decide whether there exists a group with cost value at most C. The reduction is inspired by the proof for the hardness of the *multiple-choice cover* (MCC) problem [Arkin and Hassin 2000] (that is different from our problem).

Consider a circle with diameter d and with the query point q as its center, and let each variable e_i correspond to a point in the circle, while its negation $\bar{e_i}$ corresponds to the diametrically opposite on the circle. The distance between e_i and $\bar{e_i}$ is d. We set $d = \frac{2}{3}(C + 2\epsilon)$, where $\epsilon > 0$. We also set another value $d_1 = \frac{2}{3}(C - \epsilon)$, such that $d > d_1$ and we can set ϵ to be a very small value to make sure that d is sufficiently close to d_1; thus, the distance between any two points corresponding to different variables can be no larger than d_1.

Each set S_i ($i \in [1, n]$) contains a pair of points e_i and $\bar{e_i}$, and the two points contain a distinct keyword in $q.\psi$. Each set S_j ($j \in [n+1, n+l]$) contains each triple of points corresponding to a cause C_{j-n}, and they contain a distinct keyword in $q.\psi$. Thus, to cover all keywords in $q.\psi$, a query result of q must contain one point from each S_i (i.e., e_i and $\bar{e_i}$) and must contain at least one point from each S_j (corresponding to clause C_{j-n}).

Given this mapping, we can see that if there exits a truth assignment for Φ, we can find a group χ for the MAX+MAX SGK query. In this group, all the keywords in q are covered, and the cost can be computed as $\mathsf{Cost}(q, \chi) = \max_{o \in \chi} \mathsf{Dist}(o, q) + \max_{o_i, o_j \in \chi} \mathsf{Dist}(o_i, o_j) = \frac{d}{2} + \max \mathsf{Dist}(o_i, o_j), o_i, o_j \in \chi$, which indicates that a feasible solution χ with cost at most $\frac{d}{2} + d_1 = C$ exists. On the other hand, if there exists a subset of points on the circle whose diameter is at most d covering all the query keywords, then there exists a truth assignment for the instance Φ. This completes the proof.

(3) The proof for the MAX+MAX SGK query is applicable to the MIN+MAX SGK query by replacing $\max_{o \in \chi} \mathsf{Dist}(o, q)$ with $\min_{o \in \chi} \mathsf{Dist}(o, q)$.

(4) Similar to the proof for the MAX+MAX and MIN+MAX SGK queries, we also reduce the 3-SAT problem to that of answering the SUM+MAX SGK query. According to this mapping, the group must contain exactly n point and thus, for every group, the aggregated distance of all the points to the query point (the centerpoint in the mapping) is the same. □

In the article, we consider answering the SUM, MAX+MAX, and MIN+MAX queries; the SUM+MAX query is left for future work. Given an SGK query, when there are multiple optimal groups of objects, we choose one group randomly. For ease of presentation,

we disregard parameter α in the cost functions in the rest of the article. However, the proposed algorithms remain applicable when α is enabled. The parameter affects the performance bounds of the approximation algorithms, which will be explained in later sections.

The prior discussion only focuses on finding the optimal group. However, it is attractive to be able to return several results, thus providing users with more options. Based on Definition 2.1, we define the top-k spatial group keyword query.

Definition 2.4 (*Top-k Spatial Group Keyword Query*). A *top-k spatial group keyword* (kSGK) *query* q is of the form $\langle q.\lambda, q.\psi, k\rangle$, where $q.\lambda$ and $q.\psi$ are as defined before, and k is the number of groups to be retrieved. It finds k groups of objects (possibly with common objects) $\mathcal{X}_k = \langle \chi_1, \ldots, \chi_k\rangle$, where χ_i $(1 \leq i \leq k)$ is a feasible group of q, such that any feasible group $\chi_m \notin \mathcal{X}_k$ has a cost that exceeds that of any group $\chi_i \in \mathcal{X}_k$.

We further propose to provide support for object weights in the spatial group keyword query. The object weights may come from different sources and may capture different aspects of the objects. For example, they can capture the popularity as measured by numbers of check-ins, they can be user ratings, or can be the text relevance to the query keywords. This support for object weights is relevant because users may find such ratings important, in addition to the distances to the query. For example, a user may prefer to have dinner at a further-away restaurant with a high rating rather than at closer restaurant with a low one.

To accommodate this generalization, we compute the cost of a group considering both the spatial proximity and the object weights. In particular, we compute the "weighted distance" instead of the Euclidean distance of an object to a query: $\mathsf{wDist}(o, q) = \mathsf{Wt}(o, q) \cdot \mathsf{Dist}(o, q)$, where $\mathsf{Wt}(o, q)$ computes the weight of o with respect to q (this is done as in previous work that also uses weighted distances [Aurenhammer and Edelsbrunner 1984; Wu et al. 2011]). We study the weighted SUM, MAX+MAX, and MIN+MAX SGK queries, with cost functions defined as follows.

Weighted SUM *cost function.* $\mathsf{Cost}(q, \chi) = \sum_{o\in\chi} \mathsf{wDist}(o, q)$.

Weighted MAX+MAX *cost function.* $\mathsf{Cost}(q, \chi) = \alpha \max_{o\in\chi}(\mathsf{wDist}(o, q)) + (1-\alpha) \max_{o_3\in\chi} (\mathsf{Wt}(o_3, q)) \cdot \max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1, o_2))$.

Weighted MIN+MAX *cost function.* $\mathsf{Cost}(q, \chi) = \alpha \min_{o\in\chi}(\mathsf{wDist}(o, q)) + (1-\alpha) \max_{o_3\in\chi} (\mathsf{Wt}(o_3, q)) \cdot \max_{o_1,o_2\in\chi}(\mathsf{Dist}(o_1, o_2))$

As mentioned, object weights can capture aspects of objects such as user ratings, popularity, and text relevance to the query. The larger the weight, the more attractive the object. In the following, we interpret the weight of object o as the text relevance (computed by the language model [Zhai and Lafferty 2004]) of $o.\psi$ to $q.\psi$, denoted by $\mathsf{Sim}(o, q)$. Note that using text relevance as the object weight is more challenging than using other aspects of objects such as user ratings because the weight of an object depends on the query keywords, which is not the case for, for example, ratings. We define $\mathsf{Wt}(o, q) = e^{-\mathsf{Sim}(o,q)}$ so that the object weight has the same effect on the cost of a group as does the spatial proximity (the smaller the value is, that is, the higher the rating, the lower the cost). Next, we observe that a desired group is expected to have a small diameter and to contain objects with large weights. Hence we multiply the maximum value of $\mathsf{Wt}(\cdot, \cdot)$ in a group with the group's diameter. This means that we expect the worst weight (e.g., the lowest rating) in a group to be large. It is straightforward to use instead the average value or the sum value of $\mathsf{Wt}(\cdot, \cdot)$ in the group in the cost functions.

All three weighted SGK queries are NP-hard, since the three original spatial group keyword queries are special cases of their weighted versions.

Table I. Summary of Queries and Algorithms

Query	Algorithms	Section	Complexity	Approximation Ratio
SUM	SUM-Appro	Section 4.1	$O(nm \log m)$	$\sum_{i=1}^{n} \frac{1}{i}$
	SUM-Exact	Section 4.2	$O(2^n m \log m)$	N.A.
MAX+MAX	MAXMAX-Appro1	Section 5.1	$O(m \log m)$	3
	MAXMAX-Appro2	Section 5.2	$O(\frac{m}{n}(\log \frac{m}{n} + m \log m))$	1.8
	MAXMAX-Exact	Section 5.3	$O(\frac{m}{n} r^n)$	N.A.
MIN+MAX	MINMAX-Appro	Section 6.1	$O(m \log m)$	3
	MINMAX-EXACT	Section 6.2	$O(\frac{m}{n} r^n)$	N.A.
weighted SUM	WSUM-Appro	Section 8.1.1	$O(nm \log m)$	$\sum_{i=1}^{n} \frac{1}{i}$
	WSUM-Exact	Section 8.1.2	$O(2^n m \log m)$	N.A.
weighted MAX+MAX	WMAXMAX-Appro	Section 8.2.1	$O(m \log m)$	$1 + 2e$
	WMAXMAX-Exact	Section 8.2.2	$O(\frac{m}{n} r^n)$	N.A.
weighted MIN+MAX	WMINMAX-Appro	Section 8.3.1	$O(m \log m)$	$3e^2$
	WMINMAX-Exact	Section 8.3.2	$O(\frac{m}{n} r^n)$	N.A.

We summarize the queries proposed in the article and the algorithms designed for them in Table I. In the table, n is the number of query keywords and m the number of objects that contain at least one query keyword. We use r to represent the number of objects in the search space around a candidate object, which will be explained in Sections 5.3 and 6.2.

3. PRELIMINARIES: THE IR-TREE

We briefly review the IR-tree [Cong et al. 2009; Wu et al. 2012a] which we use as an index structure in the algorithms to be presented. We note that other spatial-keyword indexes (e.g. De Felipe et al. [2008]) may be used in its place.

The IR-tree is essentially an R-tree [Guttman 1984] extended with inverted files [Zobel and Moffat 2006]. Each leaf node in the IR-tree contains entries of the form $(o, o.\lambda, o.di)$, where o refers to an object in dataset D, $o.\lambda$ is the bounding rectangle of o, and $o.di$ is an identifier of the description of o. Each leaf node also contains a pointer to an inverted file with the keywords of the objects stored in the node.

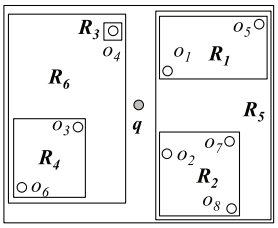
An inverted file index has two main components:

—a vocabulary of all distinct words appearing in the description of an object; and
—a posting list for each word t that is a sequence of identifiers of those objects whose descriptions contain t.

Each non-leaf node R in the IR-tree contains a number of entries of the form $(cp, cp.\lambda, cp.di)$, where cp is the address of a child node of R, $cp.\lambda$ is the *minimum bounding rectangle* (MBR) of all rectangles in entries of the child node, and $cp.di$ is an identifier of a pseudo text description that is the union of all text descriptions in the entries of the child node.

As an example, Figure 1(a) contains eight spatial objects $o_1, o_2, \ldots, o_8$, and Figure 1(b) shows the words appearing in the description of each object. Figure 2 illustrates the corresponding IR-tree, and Table II shows the content of the inverted files associated with the nodes.

In the weighted SGK queries, we use the text relevance of an object to a query as the weight of the object. Following past work [Cong et al. 2009], we utilize the language model [Zhai and Lafferty 2004] to compute the text relevance. The inverted file associated with each node also stores the term weight information. The language



(a) object locations

object	words	distance to q
o_1	t_1, t_2	1.4
o_2	t_2, t_3	2.8
o_3	t_1, t_3	3
o_4	t_1, t_5	3.2
o_5	t_2, t_4	8
o_6	t_4, t_6	9
o_7	t_4, t_5	4.5
o_8	t_1, t_5	8

(b) object descriptions

Fig. 1. A dataset of spatial keyword objects.

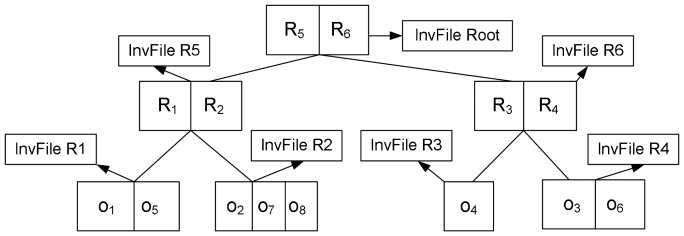


Fig. 2. Example IR-tree.

Table II. Content of Inverted Files of the IR-Tree

Root	R_5	R_6	R_1	R_2	R_3	R_4
t_1: R_5, R_6	t_1: R_1, R_2	t_1: R_3, R_4	t_1: o_1	t_1: o_8	t_1: o_4	t_1: o_3
t_2: R_5	t_2: R_1, R_2	t_3: R_4	t_2: o_1, o_5	t_2: o_2	t_5: o_4	t_3: o_3
t_3: R_5, R_6	t_3: R_2	t_4: R_4	t_4: o_5	t_3: o_2		t_4: o_6
t_4: R_5, R_6	t_4: R_1, R_2	t_5: R_3		t_4: o_7		t_6: o_6
t_5: R_5, R_6	t_5: R_2	t_6: R_4		t_5: o_7, o_8		
t_6: R_6						

model we use to compute $\mathsf{Sim}(o, q)$ is given by the following equation:

$$Sim(q, o) = \prod_{t \in q.\psi} \hat{p}(t|\theta_{o.\psi}),\ \hat{p}(t|\theta_{o.\psi}) = (1-\lambda)\frac{tf(t, o.\psi)}{|o.\psi|} + \lambda\frac{tf(t, Coll)}{|Coll|}. \quad (2)$$

Here, $tf(t, d)$ takes a term t and a document d as arguments and returns the number of occurrences of the t in d; $Coll$ is the document that consists of the collection of all documents associated with objects in $\mathcal{D}$; $tf(t, d)/|d|$ is the maximum likelihood estimate of t in d; and λ is a smoothing parameter of the Jelinek-Mercer smoothing method.

In a leaf node R of the IR-tree, the posting list of each term t is a sequence of pairs $\langle o, \hat{p}(t|\theta_{o.\psi})\rangle$, where o is that object in R whose text description contains t, and $\hat{p}(t|\theta_{o.\psi})$ is the term weight of t in $o.\psi$ as computed by Eq. (2). In a non-leaf node R, the posting list of each term t is a sequence of pairs $\langle cp, wt_{t,cp.\psi}\rangle$, where cp is the child node of R that contains t, and $wt_{t,cp.\psi}$ is the term weight of t in the pseudo document of cp (denoted by $cp.\psi$), which is the maximum weight of term t in the documents contained in the subtree rooted at node cp. The concept of a pseudo document of a node cp enables to estimate a bound of the text relevance to a query of all documents contained in the subtree rooted at cp.

4. PROCESSING SUM SPATIAL GROUP KEYWORD QUERIES

An approximation algorithm is first presented in Section 4.1. The number of keywords of a query may be small in some applications, and this motivates us to develop an exact algorithm for processing the SUM SGK query. A dynamic programming algorithm that uses an index to prune the search space is described in Section 4.2.

4.1. Approximation Algorithm

We show that the problem of answering the SUM query is NP-hard by a reduction from the *weighted set cover* (WSC) problem in Theorem 2.3. The reduction in the proof is approximation preserving. Thus the approximation properties of the WSC problem carry over to our problem.

For the WSC problem, it is known (see Chvatal [1979]) that a greedy algorithm is an H_k-approximation algorithm for the weighted k-set cover, where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k-th harmonic number. In our problem, k is the number of query keywords. Thus we can adapt the greedy algorithm to process the SUM spatial group keyword query.

In the WSC problem, there are m elements $\{e_1, \ldots, e_m\}$ and a set S of n sets $\{S_1, \ldots, S_n\}$, where each set S_i has a weight w_i. The objective is to select a group of sets from S such that they cover all elements with the smallest total weight. The basic idea of the greedy algorithm for WSC is to perform a sequence of steps where, in each step, a set S_i is selected greedily from S such that the value of its weight divided by the uncovered elements is the smallest.

A straightforward method of adapting the greedy algorithm is to decompose the given user query q dynamically into a sequence of partial queries, each containing a different set of keywords depending on the preceding partial queries, and then to evaluate these partial queries. Specifically, we start with the user query q, which can be regarded as the first partial query, and we find the object with the lowest cost that covers part or all of the keywords in q. The object is added to the result set. The uncovered keywords in q form a new partial query with the same spatial location as q. We then find an object with the lowest cost that covers part or all of the keywords in the new partial query. This process continues until all keywords are covered, or until some keyword cannot be covered by any object. This method needs to scan the dataset multiple times, once for each partial query.

To avoid multiple scans, we propose a greedy algorithm on top of the IR-tree. We proceed to focus on two aspects of the idea that are important to performance: (1) how to find that object with the lowest cost for each partial query using the IR-tree; and (2) whether we can reuse the computation for the preceding partial query when computing the next partial query.

Given a partial query q_s, we adopt the best-first strategy to traverse the IR-tree. We use a min-priority queue to maintain the intermediate results. The key of the queue is the cost of each element. The cost of an object o is computed by $\frac{\mathsf{Dist(o,q)}}{|o.\psi \cap q_s.\psi|}$; the cost of a node entry R is computed by $\frac{\mathsf{minDist(R,q)}}{|R.\psi \cap q_s.\psi|}$, where $\mathsf{minDist}(R, q)$ represents the minimum distance between q and R. This way of computing the costs of objects and nodes guarantees that our algorithm has an approximation ratio $H_k = \sum_{i=1}^{k} \frac{1}{i}$.

LEMMA 4.1. *Given a partial query q_s and an IR-tree, the cost of a node is a lower bound of the cost of any of its child nodes.*

PROOF. Given a node R and any of its child nodes R', we have $\mathsf{minDist}(R, q) \leq \mathsf{minDist}(R', q)$, and $|R.\psi \cap q_s.\psi| \geq |R'.\psi \cap q_s.\psi|$.

Lemma 4.1 says that the cost of a node is a lower bound of the costs of all objects in the subtree rooted at the node. Thus, if the cost exceeds that of some object that

has been visited, we can disregard all objects in the subtree for q_s. This guarantees the correctness of the best-first strategy for finding an object with the lowest cost for a partial query q_s.

We next discuss whether we can reuse the computation for preceding partial queries. An obvious method is to process each partial query from scratch. However, this incurs repeated computation when a node or object is visited multiple times. To avoid this, we divide the entries (corresponding to leaf and non-leaf nodes) in the priority queue into two parts: (1) those entries that have already been visited when processing previous partial queries; and (2) those not yet been visited.

LEMMA 4.2. *The elements in the priority queue that have been visited when processing previous partial queries can be disregarded when processing a new partial query.*

PROOF. The keyword set of a previous partial query is a superset of the keyword set of a new partial query. For a visited node, all its entries containing keywords of the new partial query have been enqueued into the priority queue; thus we can disregard the elements that have been visited. □

The pseudocode is outlined in Algorithm 1. The algorithm uses a min-priority queue for the best-first search with the cost as the key. Variable *mSet* keeps the keyword set of the current partial query, and *pSet* keeps the keyword set of the preceding partial query. For each partial query, we use the best-first search to find an object that overlaps with the query keyword *mSet* and has the lowest cost.

Whenever the algorithm pops an object from U, it is guaranteed that the text description of the object overlaps with *mSet* (the keyword set of the current partial query), and that the object has the lowest cost. Thus it becomes part of the result. The algorithm proceeds with the next partial query by changing the keyword component

ALGORITHM 1: SUM-Appro(q, $irTree$)

```
1  U ← new min-priority queue;
2  U.Enqueue(irTree.root, 0);
3  Group ← ∅; Cost ← 0;
4  mSet ← q.ψ; pSet ← q.ψ;
5  while mSet ≠ ∅ and U is not empty do
6      p ← U.Dequeue();
7      Cost ← Cost + p.Key;
8      if p is an object then
9          Group ← Group ∪ p;
10         pSet ← mSet ;
11         mSet ← mSet \ p.ψ;
12         for each entry p′ in U do
13             if p′.ψ ∩ p.ψ ≠ ∅ then p′.key = p′.key∗|p′.ψ∩mSet| / |p′.ψ∩pSet| ;
14             else remove p from U;
15         reorganize the priority queue U using new key values;
16     else
17         for each entry p′ in node e do
18             if mSet ∩ p′.ψ ≠ ∅ then
19                 if p is a non-leaf node then dist ← minDist(p′, q);
20                 else dist ← Dist(p′, q);
21                 U.Enqueue(p′, dist / |mSet∩o.ψ|);
22 return Cost and Group ;                                   // results
```

Table III. Example Dataset

	o_1	o_2	o_3	o_4
Distance to the query	1	2	2.5	4
Keywords	t_1,t_2	t_2,t_3	t_1,t_3	t_1

(line 11). Based on Lemma 4.2, we do not need to scan all objects to process the new partial query. Rather, we only have to update the unvisited elements in the priority queue with the new cost based on the new partial query (lines 12–15). We then use the best-first search to process the new partial query.

Assume there are n query keywords, and that the number of relevant objects is m. The number of relevant nodes is O(m), and of the elements in the queue is also O(m). In the worst case, all nodes are inserted into the queue, and the complexity of this step is O($m \log m$). Next, at most n objects are dequeued from the queue, and we also need to reorganize the queue, which costs O($nm \log m$). For the remaining objects, we only need to remove them from the queue, which costs at most O($m \log m$). Thus the total worst-case complexity is O($nm \log m$).

4.2. Exact Algorithm

A straightforward exact algorithm enumerates every subset of spatial objects whose text descriptions overlap with the query keyword set in D and check whether it covers all query keywords and has the smallest cost. This yields an exponential running time in the number of relevant objects, which is very expensive. A better method is to perform an exhaustive search on a smaller set of objects. The idea is based on the following lemma.

LEMMA 4.3. *Consider a query q and two objects o_i and o_j, each containing a subset of the query keywords. Let $ws_i = q.\psi \cap o_i.\psi$ and $ws_j = q.\psi \cap o_j.\psi$. If* $\mathsf{Dist}(o_i, q) < \mathsf{Dist}(o_j, q)$, *$\{o_i\}$ is a better group than $\{o_j\}$ for any keyword subset of $ws_i \cap ws_j$.*

PROOF. The proof is obvious since o_i always incurs lower cost than does o_j for any keyword subset $ws_i \cap ws_j$. □

Thus, given a subset of query keywords ws, among those objects covering ws, the one that is the closest to the query contributes the lowest cost to ws.

Example 4.4. Consider a query q with keywords $q.\psi = \{t_1, t_2, t_3\}$ and the four objects in Table III. We know that $\mathsf{Dist}(o_1, q) < \mathsf{Dist}(o_2, q)$ and $o_1 \cap o_2 = \{t_2\}$. According to Lemma 4.3, $\{o_1\}$ is a better result set than $\{o_2\}$ for the query with keyword set $\{t_2\}$.

Since the set of query keywords is small, the number of its subsets is not large, although exponential in the number of query keywords. For each subset of query keywords, we find the object that covers the subset of query keywords and has the lowest cost. We call the set of these objects $objSet$. We then only need to do an exhaustive search on these objects to find the best group. The time complexity of this method is O($2^{|objSet|}$). With n keywords in the keyword component of a query q, at most $(2^n - 1)$ objects need to be considered, and thus its worst time complexity is O(2^{2^n}), which means that this method is still time consuming.

As the number of query keywords may be small in many cases, we develop an exact algorithm using dynamic programming. Given a query q, for each subset X of $q.\psi$, we denote the set of objects that cover X with the smallest cost by $\mathsf{Group}(X)$, and denote the cost of covering X by $\mathsf{Cost}(X)$. The idea of the algorithm is that, given a query q, we

first initialize the cost of subsets of $q.\psi$ according to Lemma 4.3 as

$$\begin{aligned} \mathsf{Group}(X) &= \begin{cases} \arg\min_{o \in objSet \wedge X \subseteq o.\psi} \{\mathsf{Dist}(o,q)\}, & \exists o(X \subseteq o.\psi), \\ \emptyset, & \text{otherwise} \end{cases} \\ \mathsf{Cost}(X) &= \begin{cases} \min_{o \in objSet \wedge X \subseteq o.\psi} \{\mathsf{Dist}(o,q)\}, & \exists o(X \subseteq o.\psi) \\ \infty, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Next, we process the subsets of $q.\psi$ with infinite cost in ascending order of their length. Given a subset X of $q.\psi$, $\mathsf{Group}(X)$ cost must contain at least one object in $objSet$. Hence, we can check each object in $objSet$ that contains some keywords in X with its corresponding complementary keyword subset with respect to X (whose optimal cost is already known) to find the lowest cost for X, as shown in Eq. (4).

$$\mathsf{Cost}(X) = \min(\mathsf{Cost}(X), \min_{o \in objSet \wedge o.\psi \cap X \neq \emptyset} (\mathsf{Cost}(X \setminus o.\psi) + \mathsf{Dist}(o,q))). \tag{4}$$

Given a keyword subset X, the complexity of finding the lowest cost of X is $O(|objSet|)$, and hence the complexity of finding the optimal group from $objSet$ in this algorithm is $O(2^n \cdot |objSet|)$, which is much better than the simple method with complexity $O(2^{|objSet|})$.

Based on Eqs. (3) and (4), we can scan the whole dataset to find all objects relevant to the query and perform dynamic programming to find the optimal group. This has two drawbacks: (1) it wastes computation when checking many unnecessary objects that do not contain any query keyword, and (2) all those objects whose text descriptions overlap with the query keywords are scanned to obtain the lowest costs for the query keyword subsets.

To overcome the first drawback, we utilize the IR-tree that enables us to retrieve only those objects that contain some query keywords while avoiding checking those containing no query keywords. To address the second drawback, we show that it is not always necessary to scan all the objects covering part of the query keywords.

We propose the following principle for our algorithm: we process objects in ascending order of their distances to a query q. By following this order, we know that the lowest cost of a subset is always contributed by a single or a group of closer objects based on Lemma 4.3.

LEMMA 4.5. *Consider a query q. If we process objects in ascending order of their distances to q, when we reach an object o_i containing a query keyword subset ws, all subsets of ws will get their lowest costs.*

PROOF. Obvious since all objects to be visited after o_i have larger cost for any subset of ws; thus its lowest cost is either contributed by o_i or by objects visited earlier. □

Example 4.6. Recall the query in Example 4.4. We first process object o_1, and we know that 1 is the lowest cost of subsets $\{t_1, t_2\}$, $\{t_1\}$, and $\{t_2\}$. Then we reach o_2, and we know that 2 is the lowest cost of subsets $\{t_2, t_3\}$ and $\{t_3\}$ ($\{t_2\}$ already has lowest cost 1).

Based on Lemma 4.5, we can derive a stopping condition for our algorithm: that it reaches an object that contains all the query keywords. However, if no such object exists in the dataset, the algorithm is still required to scan to the furthest-away object containing some query keywords before it can stop. In the example in Table III, we need to read all four objects. But if the third furthest-away object o_3 covers $\{t_1, t_2, t_3\}$, we need not read o_4.

We proceed to present an additional stopping condition.

LEMMA 4.7. *Given two subsets ws_i and ws_j of query keywords, and with union $ws_u = ws_i \cup ws_j$, we have $\mathsf{Cost}(ws_u) \leq \mathsf{Cost}(ws_i) + \mathsf{Cost}(ws_j)$.*

PROOF. Obvious from Eq. (4) □

Based on Lemma 4.7, for any two keyword sets whose lowest costs are known, we can obtain an upper bound of the lowest-cost value for the keyword subset that is the union of the two keyword subsets. We denote the upper bound by $Cost_u$.

In our algorithm, we keep track of the upper bounds for those subsets whose costs are still unknown. Whenever we reach an object from which some keyword subset gets its lowest cost (according to Lemma 4.5), the subset, together with each of the subsets that have either lowest costs or upper bounds of cost (i.e., those keyword subsets that are covered by visited objects), is used to update the upper-bound cost values of the corresponding union of the keyword subsets.

Example 4.8. Recall Example 4.4. After object o_2 is scanned, $\{t_3\}$ gets its lowest cost 2. We can compute an upper bound for $\{t_1, t_3\}$ using the costs of $\{t_1\}$ and $\{t_3\}$, that is, $Cost_u(\{t_1, t_3\}) = \mathsf{Cost}(\{t_1\}) + \mathsf{Cost}(\{t_3\}) = 3$ (covered by o_1 and o_2). Similarly, we can also compute an upper-bound value 3 for $\{t_1, t_2, t_3\}$ by using the costs of $\{t_1, t_2\}$ and $\{t_3\}$, and this set is also covered by o_1 and o_2.

When we reach o_3, we get a lower cost of 2.5 for $\{t_1, t_3\}$(the previous upper bound of 3 is updated). Then $\{t_1, t_3\}$ are combined with $\{t_2\}$ to form $\{t_1, t_2, t_3\}$ with a cost of 3.5. Since this value exceeds its current upper bound, no update is needed.

We are ready to introduce a lemma that provides an early stopping condition for our algorithm.

LEMMA 4.9. *Suppose that we scan objects in ascending order of their distances to q. Given a keyword subset ws, when we reach object o_i, and if $\mathsf{Dist}(o_i, q) \geq Cost_u(ws)$, then $\mathsf{Cost}(ws) = Cost_u(ws)$, where $Cost_u(ws)$ is the current upper bound of ws.*

PROOF. We prove this by contradiction. If any object o_j further to q than o_i is a member of the best group, then it must have $\mathsf{Cost}(ws) \geq \mathsf{Dist}(o_j, q) \geq \mathsf{Dist}(o_i, q) \geq Cost_u(ws)$. Since $Cost_u(ws)$ cannot be smaller than $\mathsf{Cost}(ws)$, no further object will be contained in the best group. In addition, $Cost_u(ws)$ is the current minimum-cost value and thus becomes the lowest cost of ws. □

Example 4.10. Recall again Example 4.4. By following ascending order of distances, when the algorithm reaches o_4 ($\mathsf{Dist}(q, o_4) = 4$), we can conclude that $Cost_u(\{t_1, t_2, t_3\}) = 3$ is the lowest cost and that the best group is $\{o_1, o_2\}$.

The pseudocode is described in Algorithm 2. Those keyword subsets whose lowest costs are already known are stored in the variable *markedSet*, and the subsets that have upper bounds are stored in the variable *valuedSet*. The IR-tree is used for retrieving the next nearest object that covers some query keywords. We use a min-priority queue U to store the IR-tree nodes and objects, where their distances to the query are the keys.

The priority queue U is initialized to the root node of the IR-tree (line 4). We dequeue an element p from U, and we compute the keyword intersection *ks* between p and q (lines 7–8). If the keyword subset *ks* is contained in *markedSet* (whose lowest costs are known), we do not need to process p, according to Lemma 4.3 (line 9). Otherwise, we process p according to its type: (1) if p is a non-leaf index node, we check each of its child nodes, denoted by p', to see whether p' contains a keyword subset of q that is not contained in *markedSet*; if so, p' is inserted into U with its minimum distance to query q as its priority key (lines 10–12); (2) if p is a leaf node, we handle each object in p similarly to how we handle each child node in (1) (lines 13–15); (3) if p is an object, we first utilize its distance to q to move some keyword subsets from *valuedSet* to *markedSet*. Those subsets whose upper bounds are smaller than $\mathsf{Dist}(p, q)$ get their

ALGORITHM 2: SUM-ExactWIndex(q, $irTree$)

```
1  markedSet ← ∅, valuedSet ← ∅;
2  n ← |q.ψ|;
3  for i from 1 to 2^n − 1 do Cost[i] ← ∞, Group[i] ← ∅;
4  U ← new min-priority queue;
5  U.Enqueue(irTree.root, 0);
6  while U is not empty do
7      p ← U.Dequeue();
8      ks ← q.ψ ⋂ p.ψ ;
9      if ks ∉ markedSet then
10         if p is a non-leaf node then
11             foreach entry p′ in node p do
12                 if q.ψ ⋂ p′.ψ ≠ ∅ and q.ψ ⋂ p′.ψ ∉ markedSet then U.Enqueue(p′,
                       minDist(p′, q));
13         else if p is a leaf node then
14             foreach object o in leaf node p do
15                 if q.ψ ⋂ o.ψ ≠ ∅ and q.ψ ⋂ o.ψ ∉ markedSet then U.Enqueue(o,
                       Dist(o, q));
16         else                                                  // p is an object
17             foreach set S ∈ valuedSet do
18                 i ← MapToInteger(S);
19                 if Cost[i] < Dist(p, q) then
20                     if i = 2^n − 1 then                          // Lemma 4.9
21                         return Cost[2^n − 1] and Group[2^n − 1];
22                     valuedSet ← valuedSet \ S;
23                     markedSet ← markedSet ⋃ S;
24             foreach subset ss ⊆ ks do
25                 if ss ∉ markedSet then
26                     i ← MapToInteger(ss);
27                     markedSet ← markedSet ⋃ ss;
28                     if ss ∈ valuedSet then
29                         valuedSet ← valuedSet \ ss;
30                     Cost[i] ← Dist(p, q);
31                     Group[i] ← {p};
32             j ← MapToInteger(ks);
33             if j = 2^n − 1 then                                  // Lemma 4.5
34                 return Cost[2^n − 1] and Group[2^n − 1];
35             for i from 1 to 2^n − 1 do
36                 if Cost[i] = ∞ then continue;
37                 unionKey ← i|j;
38                 if unionKey = i or unionKey = j then continue;
39                 if Cost[unionKey] is ∞ then
40                     valuedSet ← valuedSet ⋃ MapToSet(unionKey);
41                 D ← Cost[i] + Dist(p, q);
42                 if Cost[unionKey] > D then
43                     Cost[unionKey] ← D;
44                     Group[unionKey] ← Group[i] ⋃ {p};
45 return Cost[2^n − 1] and Group[2^n − 1];
```

lowest costs (lines 17–23) according to Lemma 4.9. If the query keyword set $q.\psi$ is confirmed to get its lowest cost, the algorithm terminates (line 21). Then, for each subset ss of $q.\psi \cap p.\psi$, if its lowest cost is unknown (line 25), the object p constitutes the best group (Lemma 4.5) for ss. Since ss may already be covered by previously visited objects and have an upper bound of its lowest cost, we remove ss from $valuedSet$

Table IV. Results After Processing o_1

i	1	2	3	4	5	6	7
Cost	1	1	1	∞	∞	∞	∞
Group	o_1	o_1	o_1	null	null	null	null
Status	M	M	M	null	null	null	null

Table V. Results After Processing o_2

i	1	2	3	4	5	6	7
Cost	1	1	1	2	3	2	3
Group	o_1	o_1	o_1	o_2	o_1,o_2	o_2	o_1,o_2
Status	M	M	M	M	V	M	V

Table VI. Results After Processing o_3

i	1	2	3	4	5	6	7
Cost	1	1	1	2	2.5	2	3
Group	o_1	o_1	o_1	o_2	o_3	o_2	o_1,o_2
Status	M	M	M	M	M	M	V

(lines 28–29). Once $q.\psi$ gets its lowest cost, the algorithm terminates (lines 32–34). In lines 35–44, we combine the object p with those subsets that already have cost values (Lemma 4.7). In line 37, "|" is the bit-wise OR operator. If one is the subset of the other (line 39), we do not combine the two subsets; otherwise, we update the cost value for the union keyword subset (lines 42–44).

Example 4.11. Recall Table III in Example 4.4. The algorithm works as follows.

(1) After processing o_1, the result is as shown in Table IV, in which i is the integer representing a keyword subset and status "M" means that the subset is contained in *markedSet*. Table IV shows that $\{t_1\}$ ($i = 1$), $\{t_2\}$ ($i = 2$), and $\{t_1, t_2\}$ ($i = 3$) get their lowest costs and best groups.

(2) After processing o_2, the result is as shown in Table V. Except for $\{t_1, t_3\}$ and $\{t_1, t_2, t_3\}$, all the subsets obtain their lowest costs. The cost values of the two subsets are obtained by combining other subsets with known lowest cost. The status value "V" means that the subset is stored in *valuedSet*.

(3) After processing o_3, we obtain the result as shown in Table VI. Here, $\{t_1, t_3\}$ gets its lowest cost since it is covered by o_3.

(4) When we reach o_4, since its distance to the query is already larger than the currently lowest cost of $\{t_1, t_2, t_3\}$ (the only element in *valuedSet*), we do not need to process it. Set $\{t_1, t_2, t_3\}$ gets the lowest-cost value 3 and is moved to *markedSet*. We now find the best group and the lowest cost.

In our previous work [Cao et al. 2011], we also proposed an exact algorithm utilizing the IR-tree. The difference is that, in the new algorithm, we combine a newly accessed object p with each keyword subset s that already has a cost value to update the upper-bound cost for the union keyword subset of $p.\psi$ and s (lines 35–44). In the earlier algorithm, we combine each subset ts of $p.\psi$ with each keyword subset s that already has a cost value to update the upper-bound cost of the union of ts and s, which is not necessary and thus less efficient.

Assume the number of query keywords is n, and the number of relevant objects is m. The cost of maintaining the queue is $O(m \log m)$. When processing an object that is dequeued from U, we first check each of its query keyword subset, and then combine the object with existing keyword subsets that already have a cost, and this costs $O(2^n)$. Therefore the worst-case time complexity of this algorithm is $O(2^n m \log m)$.

5. PROCESSING MAX+MAX SPATIAL GROUP KEYWORD QUERIES

We present two approximation algorithms with performance bounds in Sections 5.1 and 5.2 and an exact algorithm in Section 5.3.

5.1. Approximation Algorithm 1

Given a query q, the idea of the algorithm, called MAXMAX-Appro1, is to find the nearest object for each keyword t_i in $q.\psi$. The set of all such nearest objects makes up the result set. The pseudocode, which assumes the dataset is indexed using the IR-tree, is outlined in Algorithm 3. The algorithm uses a min-priority queue U for the best-first search. In each iteration, we dequeue an element p from U. If p is an object, we push it into the result set and update the uncovered keyword subset (lines 7–10); if p is a node in the IR-tree, we insert all its child nodes that contain some uncovered keywords into U (lines 12–17). The runtime of this algorithm is linear in the number of query keywords.

ALGORITHM 3: MAXMAX-Appro1(q, $irTree$)

```
1  U ← new min-priority queue;
2  U.Enqueue(irTree.root, 0);
3  Group ← ∅; Cost ← 0;
4  uSkiSet ← q.ψ ;                                   // uncovered keywords
5  while U is not empty do
6  |  p ← U.Dequeue();
7  |  if p is an object and uSkiSet ∩ p.ψ ≠ ∅ then
8  |  |  Group ← Group ∪ {p}; Cost ← Cost + Dist(p, q) ;   // add p to result
9  |  |  uSkiSet ← uSkiSet \ p.ψ;
10 |  |  if uSkiSet = ∅ then break;
11 |  else
12 |  |  read the posting lists of p for keywords in uSkiSet;
13 |  |  foreach entry p′ in node p do
14 |  |  |  if uSkiSet ∩ p′.ψ ≠ ∅ then
15 |  |  |  |  if p is a non-leaf node then
16 |  |  |  |  |  U.Enqueue(p′, minDist(p′, q));
17 |  |  |  |  else U.Enqueue(p′, Dist(p′, q));
18 return Group and Cost;                             // results
```

Example 5.1. Consider a query $q.\psi = \{t_1, t_3, t_5\}$ and the objects shown in Figure 1. Object o_1 covering t_1 is first added to the result. Then o_2 containing t_3 is added and, when o_4 containing t_5 is retrieved, we obtain a group. Object o_4 has the maximum distance to query, which is 3.2. The maximum diameter is 6, which is the distance between o_2 and o_4. Thus the cost of this group is 9.2.

We proceed to show that MAXMAX-Appro1 is within an approximation factor of 3. We denote the group returned by MAXMAX-Appro1 as G_{app_1}, and denote the optimal group by G_{opt}.

THEOREM 5.2. *The cost of G_{app_1} for a given query q, is at most 3 times the cost of G_{opt}:* $\mathsf{Cost}(q, G_{app_1}) \leq 3 \cdot \mathsf{Cost}(q, G_{opt})$. *When α is enabled in the cost function, the ratio is* $\frac{2}{\alpha} - 1$.

PROOF. Let o_f denote the furthest-away object from q in G_{app_1}, and let $d = \mathsf{Dist}(o_f, q)$. Obviously, the optimal solution G_{opt} satisfies $\mathsf{Cost}(q, G_{opt}) \geq d$. In the group G_{app_1}, the largest possible distance between two objects in G_{app_1} is $2d$. Thus we have the following cost: $\mathsf{Cost}(q, G_{app_1}) \leq d + 2d \leq 3 \cdot \mathsf{Cost}(q, G_{opt})$.

When α is enabled in the cost function, the optimal solution G_{opt} satisfies $\mathsf{Cost}(q, G_{opt}) \geq \alpha d$, and $\mathsf{Cost}(q, G_{app_1}) \leq \alpha d + (1-\alpha)2d$; we thus obtain $\frac{\mathsf{Cost}(q, G_{app_1})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{2-\alpha}{\alpha}$. □

Assume that there are n query keywords and m relevant objects. Thus the number of relevant nodes is O(m), and the size of the queue is O(m) as well. In the worst case, before we can get an object from the queue, all relevant nodes are inserted into the queue, which costs O($m \log m$). At most n objects are dequeued from the queue to form the result group, and the others are removed directly from the queue. This step costs O($m \log m$) in the worst case. Hence the complexity of this algorithm is O($m \log m$) in the worst case.

5.2. Approximation Algorithm 2

Based on MAXMAX-Appro1, we present an algorithm with a better approximation bound.

The underlying idea can be described as follows. We first invoke the algorithm MAXMAX-Appro1 to find a group of objects G_{app_1}. Let t_{inf} be the most infrequent keywords in $q.\psi$. Next, for each object o_i containing t_{inf}, we create a new query q_{o_i} using the position of o_i and the keywords of the original query q, that is, $q_{o_i}.\lambda = o_i.\lambda$ and $q_{o_i}.\psi = q.\psi \setminus o_i.\psi$. We then invoke MAXMAX-Appro1 to find a group of objects for q_{o_i}, denoted by G_{o_i}, and we compute the cost of this group with respect to q. After each object containing t_{inf} is processed, we find the group with the smallest cost, and compare it with G_{app_1}. Finally, we return the one with smaller cost as the result.

We preprocess the dataset to compute the frequency of each keyword, that is, the number of objects that contain a given keyword. Before executing the algorithm, we load a file that contains the keyword frequencies into memory and organize the keywords with their frequencies in a hash map. Given a query q, we read the frequency of each keyword in $q.\psi$ from the hash map to find t_{inf}.

This algorithm only focuses on those objects containing the most infrequent query keyword. However, the number of such objects may still be large. We show that it is not necessary to process each object containing t_{inf} according to the following lemma.

LEMMA 5.3. *Given a* MAX+MAX *SGK query q and the current best cost $curCost$, any object whose distance to q exceeds $curCost$ cannot be contained in the optimal group of q.*

PROOF. If an object o with distance to q larger than $curCost$ is contained in a group G, we have $\mathsf{Cost}(q, G) \geq \max_{o_1, o_2 \in G} \mathsf{Dist}(o_1, o_2) \geq \mathsf{Dist}(o, q) > curCost$, and thus G cannot be the optimal group. □

Based on Lemma 5.3, we can process the objects containing t_{inf} in ascending order of their distances to the query q. When we obtain a group, we update the current best cost if it exceeds the cost of the new found group. When we reach an object whose distance is even larger than the current best cost, we can stop and return.

The pseudocode is given in Algorithm 4. Using MAXMAX-Appro1, we first find a group that serves as the current best group (line 3). Then we find the word t_{inf} that is the most infrequent query keyword (line 4). In lines 5–20, we incrementally search for the next nearest objects containing t_{inf} within the range of $curCost$. We dequeue an element p from U in each step. If it is an IR-tree node, we check whether its minimum distance exceeds $curCost$ (line 9). If so, the algorithm terminates according to Lemma 5.3. Otherwise, we read all its child nodes and insert those nodes that contain t_{inf} into U according to their minimum distances to q (lines 9–13). If p is an object, we also compare its distance to q with $curCost$ to determine whether the algorithm terminates

ALGORITHM 4: MAXMAX-Appro2($q, irTree$)

```
1  U ← new min-priority queue;
2  U.Enqueue(irTree.root, 0);
3  (curGroup, curCost) ← MAXMAX-Appro1(q, irTree);
4  t_inf ← the most infrequent keyword in q.ψ;
5  while U is not empty do
6      p ← U.Dequeue();
7      if p is not an object then
8          if minDist(p, q) ≥ curCost then break;                    // Lemma 5.3
9          foreach entry p' in node p do
10             if t_inf ∈ p'.ψ then
11                 if p is a leaf node then
12                     U.Enqueue(p', Dist(p', q));
13                 else U.Enqueue(p', minDist(p', q));
14     else
15         if Dist(p, q) ≥ curCost then break;                       // Lemma 5.3
16         q_p.λ ← p.λ; q_p.ψ ← q.ψ \ p.ψ;
17         (group, cost) ← MAXMAX-Appro1(q_p, sirTree);
18         if cost < curCost then
19             curCost ← cost;
20             curGroup ← group;
21 return curGroup and curCost;
```

(line 15). We create a new query q_p with the position of p and the text of q, and we then find a group using q_p as the query and compute its cost (lines 16–17). If this new cost is smaller than $curCost$, we update $curCost$ and the current best group $curGroup$ (lines 18–20). Finally, we return $curGroup$ and $curCost$.

Example 5.4. Recall query q and the dataset in Example 5.1. Algorithm MAXMAX-Appro1 is first invoked to return a group $\{o_1, o_2, o_4\}$ with cost 9.2. In the query, t_3 comprises the most infrequent keywords, and is only contained in o_2 and o_3. We search for a group near o_2, that is $\{o_2, o_7, o_8\}$ with cost 10.2 ($\mathsf{Dist}(o_8, q) + \mathsf{Dist}(o_2, o_7) = 8 + 2.2$), which is worse than that of the previous one. Then we search for a group near o_3, and we find $\{o_3, o_4\}$ with cost 8.2 ($\mathsf{Dist}(o_4, q) + \mathsf{Dist}(o_3, o_4) = 3.2 + 5$). Therefore $\{o_3, o_4\}$ becomes the current best group, and we return it as the result.

We proceed to study the approximation ratio of the algorithm. We denote the group returned by MAXMAX-Appro2 as G_{app_2}.

LEMMA 5.5. *Given a query q and an object o_i containing t_{inf}, the cost of the group found at the position of o_i, that is, $\mathsf{Cost}(q, G_{o_i})$, is in the following range: $\mathsf{Dist}(o_i, q) + \mathsf{Dist}(o_i, o_{max}) \le \mathsf{Cost}(q, G_{o_i}) \le \mathsf{Dist}(o_i, q) + 3\mathsf{Dist}(o_i, o_{max})$, where o_{max} is the furthest-away object from o_i in G_{o_i}.*

PROOF. (1) $\mathsf{Dist}(o_i, q)$ is a lower bound on the distance of this group to q, and $\mathsf{Dist}(o_i, o_{max})$ is a lower bound on the diameter of this group. As a result, the minimum cost of G_{o_i} is $\mathsf{Dist}(o_i, q) + \mathsf{Dist}(o_i, o_{max})$.

(2) $\mathsf{Dist}(o_i, q) + \mathsf{Dist}(o_i, o_{max})$ is the maximum possible distance to q of G_{o_i}. Further, the diameter will not exceed $2\mathsf{Dist}(o_i, o_{max})$ since every object of G_{o_i} is in the circle with center o_i and radius $\mathsf{Dist}(o_i, o_{max})$. Therefore the cost of this group is upper bounded by $\mathsf{Dist}(o_i, q) + 3\mathsf{Dist}(o_i, o_{max})$. □

THEOREM 5.6. *The approximation ratio of algorithm MAXMAX-Appro2 is not larger than 1.8, that is, $\mathsf{Cost}(q, G_{app_2}) \le 1.8 \cdot \mathsf{Cost}(q, G_{opt})$.*

PROOF. Let o_f denote the furthest-away object from q in G_{app_1}, let o_j denote the object containing the word t_{inf} in the optimal group G_{opt}, and let o_{max} be the object that is the furthest away from o_j in G_{o_j}.

(1) Consider the case where $\mathsf{Dist}(o_j, q) \geq \mathsf{Dist}(o_f, q)$.

Object o_{max} must contain some keyword, denoted by t, which is not covered by the other objects in G_{o_j}. Among objects containing t, o_{max} is the closest to o_j. Therefore, in G_{opt}, the object covering t cannot be closer to o_j than o_{max}. As a result, $\mathsf{Dist}(o_j, o_{max})$ is a lower bound on the diameter of G_{opt}. $\mathsf{Dist}(o_j, q)$ is a lower bound on the distance between q and G_{opt}. Hence we get $\mathsf{Cost}(q, G_{opt}) \geq \mathsf{Dist}(o_j, q) + \mathsf{Dist}(o_j, o_{max})$.

Because G_{app_2} is either the group with the smallest cost among groups found on each object containing t_{inf} or G_{app_1}, we know that $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{o_j})$ and $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{app_1})$. According to Lemma 5.5, we get $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{o_j}) \leq \mathsf{Dist}(o_j, q) + 3\mathsf{Dist}(o_j, o_{max})$ and, according to Theorem 5.2, we get $\mathsf{Cost}(q, G_{app_2}) \leq 3\mathsf{Dist}(o_f, q) \leq 3\mathsf{Dist}(o_j, q)$. Thus it holds true that

$$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{\mathsf{Dist}(o_j, q) + 3\mathsf{Dist}(o_j, o_{max})}{\mathsf{Dist}(o_j, q) + \mathsf{Dist}(o_j, o_{max})}, \frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{3\mathsf{Dist}(o_j, q)}{\mathsf{Dist}(o_j, q) + \mathsf{Dist}(o_j, o_{max})}.$$

If $\mathsf{Dist}(o_j, o_{max}) \leq \frac{2}{3}\mathsf{Dist}(o_j, q)$, then $\mathsf{Dist}(o_j, q) + 3\mathsf{Dist}(o_j, o_{max}) \leq 3\mathsf{Dist}(o_j, q)$. Hence we can use the first inequality to conclude that $\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq 1.8$. Otherwise, $\mathsf{Dist}(o_j, o_{max}) > \frac{2}{3}\mathsf{Dist}(o_j, q)$, then $\mathsf{Dist}(o_j, q) + 3\mathsf{Dist}(o_j, o_{max}) > 3\mathsf{Dist}(o_j, q)$, and we can use the second inequality to prove $\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq 1.8$.

(2) Now consider the case where $\mathsf{Dist}(o_j, q) < \mathsf{Dist}(o_f, q)$.

Here, o_f must contain some keyword t that is not covered by any other objects in G_{app_1}. Since o_f is the closest object to q containing t, the object containing t in G_{opt} must be further to q than o_f, and thus $\mathsf{Dist}(o_f, q)$ is the lower bound of the maximum distance of an object in G_{opt} to q. $\mathsf{Dist}(o_j, o_{max})$ is the lower bound of the maximum distance between any pair. Therefore we can get $\mathsf{Cost}(q, G_{opt}) \geq \mathsf{Dist}(o_f, q) + \mathsf{Dist}(o_j, o_{max})$.

Because $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{o_j})$ and $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{app_1})$, we get $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{o_j}) \leq \mathsf{Dist}(o_j, q) + 3\mathsf{Dist}(o_j, o_{max}) < \mathsf{Dist}(o_f, q) + 3\mathsf{Dist}(o_j, o_{max})$ and $\mathsf{Cost}(q, G_{app_2}) \leq \mathsf{Cost}(q, G_{app_1}) \leq 3\mathsf{Dist}(o_f, q)$. Thus it holds true that

$$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{\mathsf{Dist}(o_f, q) + 3\mathsf{Dist}(o_j, o_{max})}{\mathsf{Dist}(o_f, q) + \mathsf{Dist}(o_j, o_{max})},$$
$$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{3\mathsf{Dist}(o_f, q)}{\mathsf{Dist}(o_f, q) + \mathsf{Dist}(o_j, o_{max})}.$$

$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})}$ can similarly be shown to be no larger than 1.8. Thus we complete the proof. □

THEOREM 5.7. *The approximation ratio of algorithm MAXMAX-Appro2 is not larger than* $\frac{(2-\alpha)^2}{(1-\alpha)^2+1}$ *when* α *is enabled in the cost function* ($\alpha < 1$).

PROOF. If α is enabled in the cost function, Lemma 5.5 gives $\alpha\mathsf{Dist}(o_i, q) + (1 - \alpha)\mathsf{Dist}(o_i, o_{max}) \leq \mathsf{Cost}(q, G_{o_i}) \leq \alpha\mathsf{Dist}(o_i, q) + (2 - \alpha)\mathsf{Dist}(o_i, o_{max})$.

We still let o_f denote the furthest-away object from q in G_{app_1}, let o_j denote the object containing the word t_{inf} in the optimal group G_{opt}, and let o_{max} be the object that is the

furthest away from o_j in G_{o_j}. When $\mathsf{Dist}(o_j, q) \geq \mathsf{Dist}(o_f, q)$, we obtain

$$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{\alpha\mathsf{Dist}(o_j, q) + (2-\alpha)\mathsf{Dist}(o_j, o_{max})}{\alpha\mathsf{Dist}(o_j, q) + (1-\alpha)\mathsf{Dist}(o_j, o_{max})},$$
$$\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{(2-\alpha)\mathsf{Dist}(o_j, q)}{\alpha\mathsf{Dist}(o_j, q) + (1-\alpha)\mathsf{Dist}(o_j, o_{max})}.$$

If $\mathsf{Dist}(o_j, o_{max}) \leq \frac{2-2\alpha}{2-\alpha}\mathsf{Dist}(o_j, q)$, we use the first inequality; otherwise, we use the second inequality, which gives $\frac{\mathsf{Cost}(q, G_{app_2})}{\mathsf{Cost}(q, G_{opt})} \leq \frac{(2-\alpha)^2}{(1-\alpha)^2+1}$.

When $\mathsf{Dist}(o_j, q) < \mathsf{Dist}(o_f, q)$, similar to the proof of Theorem 5.6, we obtain the same result. □

Assume there are n query keywords, and that the number of relevant objects is m. Since we only invoke MAXMAX-Appro1 around objects containing t_{inf}, the number of elements in the queue is at most $O(\frac{m}{n})$. Thus the complexity of maintaining the queue is $O(\frac{m}{n}\log\frac{m}{n})$. On each object containing t_{inf}, we invoke MAXMAX-Appro1, and cost is $O(\frac{m}{n}m\log m)$, where $O(m\log m)$ is the complexity of MAXMAX-Appro1. Hence the worst-case complexity of this algorithm is $O(\frac{m}{n}(\log\frac{m}{n} + m\log m))$.

5.3. Exact Algorithm

It is challenging to develop an exact algorithm for MAX+MAX SGK queries, as it appears that an exact algorithm cannot avoid an exhaustive search of the object space. We extend the idea of the MAXMAX-Appro2 algorithm to devise the exact algorithm.

Because the optimal group must contain an object containing the most infrequent query keyword t_{inf}, we can do an exhaustive search around each object containing t_{inf} to find the best group containing this object. We repeat this until all objects containing t_{inf} are processed. Then the group with the smallest cost must be the optimal answer to the query.

We first utilize MAXMAX-Appro2 to derive an upper-bound cost for the optimal group. It is initially used to bound the search around an object containing t_{inf}. We also develop several pruning strategies to reduce the enumeration of groups during the exhaustive search around an object. The upper-bound cost keeps decreasing as more groups are enumerated, and is used to limit the enumeration in further search. With these efforts, we expect the exact algorithm to be reasonably efficient when the dataset contains at most tens of thousands of objects and the number of query keywords is small.

I. Bounding the exhaustive space around an object containing t_{inf}. Since enumerating the groups runs exponentially with the number of objects in the search space, how to reduce the exhaustive search around an object containing t_{inf} is crucial. We proceed to explain with the help of the current best cost $curCost$.

LEMMA 5.8. *Given a* MAX+MAX *SGK query q and two objects o_i and o_j, the lower-bound value of the cost of a group G containing o_i and o_j can be computed by* $\max(\mathsf{Dist}(o_i, q), \mathsf{Dist}(o_j, q)) + \mathsf{Dist}(o_i, o_j)$. *We denote this value as $LOW_q(o_i, o_j)$.*

PROOF. $\max(\mathsf{Dist}(o_i, q), \mathsf{Dist}(o_j, q)) \leq \max_{o\in G}\mathsf{Dist}(o, q)$ and $\mathsf{Dist}(o_i, o_j) \leq \max_{o_1,o_2\in G}\mathsf{Dist}(o_1, o_2)$, and thus $LOW_q(o_i, o_j) \leq \mathsf{Cost}(q, G)$. □

Hence, if $LOW_q(o_i, o_j)$ exceeds $curCost$, we know that o_i and o_j cannot contribute to the optimal group together. We denote by C_o^r the circular region with o as the center and with r as the radius, and we denote by E_{o_1,o_2}^r the ellipse region with o_1 and o_2 as the foci and with r as the transverse diameter. We have the following lemma to find the exhaustive search space around an object containing t_{inf}.

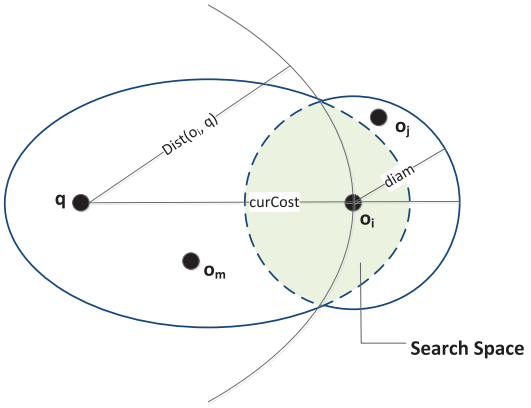


Fig. 3. Exhaustive search space around o_i containing t_{inf}.

LEMMA 5.9. *Given a* MAX+MAX *SGK query q and an object o_i containing t_{inf}, the most infrequent keyword in $q.\psi$, the search space around o_i is $C_{o_i}^{diam} \cap E_{q,o_i}^{curCost}$, where $diam = curCost - \mathsf{Dist}(q, o_i)$. We denote this region by $\mathbb{S}_{o_i}^{q}$.*

PROOF. (1) Given q and o_i, if an object o_m whose distance to o_i exceeds $diam$, then $LOW_q(o_i, o_m) \geq \mathsf{Dist}(q, o_i) + diam = curCost$. Thus any object that is possible to combine with o_i to form the optimal group must be in $C_{o_i}^{diam}$.

(2) Within $C_{o_i}^{diam}$, if an object o_j whose distance to q exceeds that of o_i then $LOW_q(o_i, o_j) = \mathsf{Dist}(q, o_j) + \mathsf{Dist}(o_i, o_j)$. Hence, if it is possible that o_i and o_j can contribute to the optimal group together, it must be true that $LOW_q(o_i, o_j) \leq curCost$, which means that o_j is in $E_{q,o_i}^{curCost}$.

Therefore we can conclude that $C_{o_i}^{diam} \cap E_{q,o_i}^{curCost}$ is the search space around o_i. Figure 3 illustrates the proof. □

Based on this lemma, we can prune objects according to the following lemma.

LEMMA 5.10 (PRUNING STRATEGY). *Given a* MAX+MAX *SGK query q and an object o if, for each object o_i containing t_{inf}, o is not in $\mathbb{S}_{o_i}^{q}$, then o can be pruned.*

PROOF. The optimal group must contain one object containing t_{inf}. If o cannot contribute to the optimal group with any object containing t_{inf} together, we know it cannot be contained in the optimal group. □

This strategy can be extended to prune an MBR in the IR-tree index.

LEMMA 5.11 (PRUNING STRATEGY). *Given an MBR R, if, for each object o in O_{inf}, R does not intersect with $\mathbb{S}_{o}^{q}$, then R can be pruned.*

PROOF. It is obviously correct based on Lemma 5.10. □

II. Enumerating the best group in the search space. Next, we cover how we do the exhaustive search within the search space around an object containing t_{inf} to find the group with the smallest cost containing this object. We call this object the *pivot* in the enumeration.

We adopt the depth-first search strategy to do the enumeration. We use *selectedSet* to store those objects that are already selected in the current enumeration. The pivot is always in *selectedSet* since it must be contained in the generated group. We use

candidateSet to store the objects within $\mathbb{S}^q_{pivot}$ (the search space of the pivot with respect to q) that are possible to combine with objects in *selectedSet* to form a group whose cost is smaller than the current best group.

The first part of the cost function of the MAX+MAX SGK query is determined by the furthest-away object from q in a group. Based on this fact, we first choose an object o_m to be the furthest-away object and put it in *selectedSet*. Then, in the following search, an object o_j is disregarded if $\mathsf{Dist}(o_j, q) > \mathsf{Dist}(o_m, q)$. This means that *candidateSet* only contains objects that are closer to q than is o_m. This greatly reduces the search space.

Then we append new objects from *candidateSet* to *selectedSet*; this step is performed iteratively. An appended object must cover a query keyword that is not already covered by the objects in *selectedSet*. The level of this depth-first search is equal to the number of query keywords, because each object contains at least one new query keyword. The current best cost *curCost* is updated when a group covering all the query keywords with smaller cost is found.

Function enumerateBestGroup($\mathbb{S}$, $pivot$, $curCost$, q)

```
1  objList ← objects in 𝕊 in ascending order of the distances to q;
2  candidateSet ← ∅;
3  for each object o in objList do
4  |   candidateSet ← candidateSet ∪ {o};
5  |   selectedSet ← {pivot, o};
6  |   pairDist ← Dist(pivot, o);
7  |   furDist ← max(Dist(pivot, q), Dist(o, q));
8  |   if candidateSet.ψ = q.ψ then
9  |   |   (cost, group) ← search(q, curCost, selectedSet, candidateSet, pairDist,
   |   |   furDist, o.di);
10 |   |   if cost < curCost then curCost ← cost; curGroup ← group;
11 return curCost and curGroup;
```

The pseudocode is described in Function enumerateBestGroup. The first step is to determine the furthest-away object. We process the objects in ascending order of their distances to the query q (line 1). In line 4, each object is added to *candidateSet* one by one to make sure that the last added object must be the furthest-away from the query location. Set *selectedSet* initially contains only $pivot$ and the furthest-away object we chose, and hence *pairDist* and *furDist* are initialized correspondingly (lines 5–7). When *candidateSet* can cover all the query keywords, we call the function *search*() to search for the best solution containing $pivot$ and o iteratively. If the new group found is better than the current group, it becomes the current best one (lines 8–10).

Although we can bound the search space according to Lemma 5.9 given a pivot, unfortunately, if the number of candidate objects in the search space is still large, the time cost of enumerating the best group prohibitively increases. We develop several pruning strategies based on both textual and geometric properties to reduce the enumeration in the function *search*().

Textual Pruning. A new object is added to the selected object set in each search step, and $selectedSet.\psi$ increases until it covers all the query keywords. Therefore, if an object o cannot contribute any new keyword to the selected objects set, this object can be ignored in the current search process. This means that it should not be inserted into *candidateSet* for the current *selectedSet*.

Distance Pruning. Since the maximum distance to the query is already fixed (represented by *furDist*), the cost of any feasible solution is only determined by the second

part in the cost function. Hence, if, after an object is added to *selectedSet*, the cost of *selectedSet* exceeds $curCost$, then the current search process can be terminated because further search can obtain no better groups.

Termination Condition. We can utilize the textual information to decide whether the current search can be terminated. If the objects in *candidateSet* cannot cover those keywords that have not been covered by *selectedSet*, we can stop since, even if we select all the objects in *candidateSet*, a group covering all the query keywords cannot be generated. Formally, if $selectedSet.\psi \cup candidateSet.\psi \not\supseteq query.\psi$, we stop the current search step.

Function search($q, curCost, selectedSet, candidateSet, pairDist, furDist, startId$)

```
1  if selectedSet.ψ = q.ψ then
2      if furDist + pairDist < curCost then
3          curCost ← furDist + pairDist;
4          curGroup ← selectedSet;
5      return curCost and curGroup;
6  nextcandSet ← ∅;
7  leftKeywords ← ∅;
8  for each candidate object o_c in candidateSet do
9      if o_c.ψ ⊆ selectedSet.ψ then continue;
10     if o_c.di < startId then continue;
11     selectedDiam ← 0;
12     for each selected object o_s in selectedSet do
13         selectedDiam ← max(selectedDiam, Dist(o_s, o_c));
14     if selectedDiam + furDist > curCost then continue;
15     nextcandSet ← nextcandSet ∪ {o_c};
16     leftKeywords ← leftKeywords ∪ o_c.ψ;
17 if leftKeywords ∪ selectedSet.ψ ≠ query.ψ then
18     return curCost and curGroup;
19 for each object o_n in nextcandSet do
20     selectedSet ← selectedSet ∪ {o_n};
21     pairDist ← max(pairDist, selectedDiam);
22     (cost, group) ← search(q, curCost, selectedSet, nextcandSet, pairDist, furDist, o_n.di);
23     if cost < curCost then
24         curCost ← cost; curGroup ← group;
25     selectedSet ← selectedSet \ o_n;
```

The pseudocode is described in Function search. First, if *selectedSet* already covers all the query keywords, we compare this group with the current best group, and return the better one as the result (lines 1–5). Then we begin the depth-first search, and append each object from *candidateSet* to *selectedSet* (lines 8–16). The textual pruning strategy is shown in line 9. Line 10 is used to avoid a duplicate enumeration of the same group. The distance pruning strategy is shown in lines 11–14. The termination condition is checked in lines 17–18. Next, after we have appended a new object to *selectedSet*, we call the function recursively to find those groups with the new candidate objects set $newcandSet$ and update $curCost$ and $curGroup$ correspondingly (lines 19–25).

III. The final exact algorithm for the MAX+MAX SGK query. The exact algorithm is presented in Algorithm 5. We use those IR-tree to find the objects containing the most infrequent keyword t_{inf}, and when the distance to q exceeds $curCost$, we terminate the algorithm. After we get an object containing t_{inf}, we first obtain its search range

according to Lemma 5.9 utilizing the IR-tree (lines 16–25). Next, we call Function enumerateBestGroup to find the best group containing this object (line 26), and we use it to update the current best group (lines 27–29). Finally, we return *curCost* and *curGroup* (line 30).

ALGORITHM 5: MAXMAX-Exact(q, $irTree$)

```
1  U ← new min-priority queue;
2  U.Enqueue(irTree.root, 0);
3  (curGroup, curCost) ← MAXMAX-Appro2(q, irTree);
4  t_inf ← the most infrequent keyword in q.ψ;
5  while U is not empty do
6      p ← U.Dequeue();
7      if p is not an object then
8          if minDist(p, q) ≥ curCost then break;
9          foreach entry p′ in node p do
10             if t_inf ∈ p′.ψ then
11                 if p is a leaf node then
12                     U.Enqueue(p′, Dist(p′, q));
13                 else U.Enqueue(p′, minDist(p′, q));
14     else
15         if Dist(p, q) ≥ curCost then break;
16         S ← ∅;
17         W ← new min-priority queue;
18         W.Enqueue(irTree.root,0);
19         while W is not empty do
20             p′ ← W.Dequeue();
21             if p′.Key > curCost then break;
22             if p′ is a node then
23                 foreach node n in p′ do
24                     if n.ψ ∩ q.ψ ≠ ∅ then W.Enqueue(n,LOW_q(p, p′));
25             else S ← S ∪ p′;
26         (group, cost) ← enumerateBestGroup(S, p, curCost, q);
27         if cost < curCost then
28             curCost ← cost;
29             s curGroup ← group;
30 return curGroup and curCost;
```

Assume there are n query keywords, and that the number of relevant objects is m. This algorithm performs the exhaustive search around each object containing the most infrequent keyword, and there are at most $O(\frac{m}{n})$ such objects. Assume that the number of objects in the search range around an object is r in the worst case. The complexity of this algorithm is $O(\frac{m}{n}r^n)$. Note that r is usually much smaller than m, and this is why the algorithm outperforms the one in the previous work [Cao et al. 2011] significantly, whose complexity is $O(m^n)$.

6. PROCESSING MIN+MAX SPATIAL GROUP KEYWORD QUERIES

6.1. Approximation Algorithm

We can use the algorithm MAXMAX-Appro1 to find a group for a given MIN+MAX SGK query. That is, we find the nearest object for each query keyword, and the set of these objects is returned as the result. The only difference is the way of computing the cost of this group. We denote this algorithm by MINMAX-Appro. We show that MINMAX-Appro is within an approximation factor of 3 as well.

THEOREM 6.1. *Given a* MIN+MAX *SGK query q, the cost of the group G_{app} returned by MAXMAX-Appro1 is at most 3 times the cost of the optimal group G_{opt}.*

PROOF. Let o_f denote the furthest-away object from q in G_{app}, and let $d = \mathsf{Dist}(o_f, q)$. For the solution G_{app}, the largest possible distance between two objects in G_{app} is $2d$, and the largest possible minimum distance between an object in G_{app} and q is d. Object o_f must contain some keyword t that is not covered by any other object in G_{app}. The optimal group must contain an object containing t. Since o_f is the nearest one with respect to t, G_{opt} must contain an object whose distance to q is no smaller than d. Denote o_n as the nearest object to q and o_t as the object containing t in G_{opt}. The maximum distance between any two objects in G_{opt} must be no smaller than $\mathsf{Dist}(o_n, o_t)$. Thus we can obtain $\mathsf{Cost}(G_{opt}, q) \geq \mathsf{Dist}(o_n, q) + \mathsf{Dist}(o_n, o_t) \geq \mathsf{Dist}(o_t, q) \geq d$, according to the triangle inequality. Finally, we have $\mathsf{Cost}(G_{app}, q) \leq d + 2d \leq 3 \cdot \mathsf{Cost}(G_{opt}, q)$. □

THEOREM 6.2. *When α is enabled in the cost function, the approximation ratio of MINMAX-Appro is $\frac{2}{\alpha} - 1$ when $\alpha < 0.5$ and $\frac{2-\alpha}{1-\alpha}$ when $\alpha \geq 0.5$.*

PROOF. When α is enabled in the cost function, matters are more complex. Let o_f be the furthest-away object in G_{app}, let o_n be the nearest object in G_{opt}, and let o_t be the object that contains the same query keyword as does o_f in G_{opt}. Hence it is true that $\mathsf{Dist}(o_n, q) \leq \mathsf{Dist}(o_t, q)$ and that $\mathsf{Dist}(o_f, q) \leq \mathsf{Dist}(o_t, q)$.

It is obvious that $\mathsf{Cost}(G_{app}, q) \leq \alpha\mathsf{Dist}(o_f, q) + 2(1-\alpha)\mathsf{Dist}(o_f, q) = (2-\alpha)\mathsf{Dist}(o_f, q)$. For the optimal group, $\mathsf{Cost}(G_{opt}, q) \geq \alpha\mathsf{Dist}(o_n, q) + (1-\alpha)\mathsf{Dist}(o_n, o_t) \geq (2\alpha - 1)\mathsf{Dist}(o_n, q) + (1-\alpha)(\mathsf{Dist}(o_n, q) + \mathsf{Dist}(o_n, o_t))$. According to the triangle inequality, $\mathsf{Cost}(G_{opt}, q) \geq (2\alpha - 1)\mathsf{Dist}(o_n, q) + (1-\alpha)\mathsf{Dist}(o_t, q)$.

When $\alpha \geq 0.5$, the smallest value of $\mathsf{Dist}(o_n, q)$ is zero, and thus $\mathsf{Cost}(G_{opt}, q) \geq (1-\alpha)\mathsf{Dist}(o_t, q) \geq (1-\alpha)\mathsf{Dist}(o_f, q)$, and we have $\frac{\mathsf{Cost}(G_{app}, q)}{\mathsf{Cost}(G_{opt}, q)} \leq \frac{2-\alpha}{1-\alpha}$. When $\alpha < 0.5$, since $\mathsf{Dist}(o_n, q)$ is at most $\mathsf{Dist}(o_t, q)$, we know that $\mathsf{Cost}(G_{opt}, q) \geq \alpha\mathsf{Dist}(o_t, q) \geq \alpha\mathsf{Dist}(o_f, q)$, and thus $\frac{\mathsf{Cost}(G_{app}, q)}{\mathsf{Cost}(G_{opt}, q)} \leq \frac{2-\alpha}{\alpha}$. □

6.2. Exact Algorithm

It is challenging to develop an exact algorithm for MIN+MAX SGK queries as well, since exhaustive search in the object space is also required. We follow the idea of the MAXMAX-Exact algorithm to design the exact algorithm for the MIN+MAX SGK query.

We first utilize the MINMAX-Appro algorithm to derive an upper bound for the optimal group of a given MIN+MAX SGK query q. Then we process those objects containing the most infrequent query keyword t_{inf} in ascending order of their distances to the query q. We still call such an object a *pivot*. Around each pivot, we first bound the exhaustive search space, that is, we find all those objects with which it is possible to form a group whose cost is smaller than that of the current best group with the pivot. Then, within this search space, we do an exhaustive search similar to that in MAXMAX-Exact to find the best group containing the pivot, and update the current best cost correspondingly.

I. Bounding the exhaustive search space around an object containing t_{inf}. Recall that, in the MAXMAX-Exact algorithm, given a pivot object o_i, the maximum distance to q of a group containing o_i must be larger than the distance of o_i to q. However, $\mathsf{Dist}(o_i, q)$ cannot bound the minimum distance to q of a group containing o_i, which could even be zero. Thus the search space around a pivot in the MIN+MAX SGK query is different from that of the MAX+MAX SGK query.

LEMMA 6.3. *Given a* MIN+MAX *SGK query q and an object o_i containing t_{inf}, the most infrequent keyword in $q.\psi$, the search space around o_i is $C_{o_i}^{curCost} \cap C_q^{curCost}$, where*

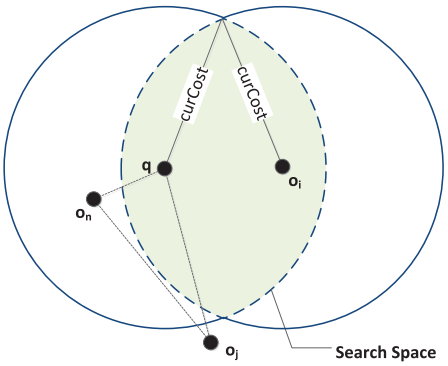


Fig. 4. Exhaustive search space around o_i containing t_{inf}.

curCost is the current best cost. With a slight abuse of notation, we denote this area by $\mathbb{S}^q_{o_i}$ *as well.*

PROOF. Denote a group containing both o_i and o_j as G_{o_i,o_j}. First, if an object o_j whose distance to o_i is larger than $curCost$, then G_{o_i,o_j} must have cost larger than $curCost$. Thus the candidate objects must be within the range of $C^{curCost}_{o_i}$. Second, denote the closest object to q in G_{o_i,o_j} as o_n. Then $\mathsf{Cost}(G_{o_i,o_j}, q) = \mathsf{Dist}(o_n, q) + \max_{r_1,r_2\in\chi}(\mathsf{Dist}(r_1, r_2)) \geq \mathsf{Dist}(o_n, q) + \mathsf{Dist}(o_n, o_j) \geq \mathsf{Dist}(o_j, q)$. Hence, if G_{o_i,o_j} has smaller cost than does $curCost$, then o_j must have distance to q smaller than $curCost$. Therefore o_j is in the range of $C^{curCost}_{o_i} \cap C^{curCost}_{q}$. Figure 4 illustrates the proof. □

II. Enumerating the best group in the search space. We introduce how we do the exhaustive search within the search space around the pivot for a given MIN+MAX SGK query.

This step is similar to that of the MAXMAX-Exact algorithm. We adopt the depth-first search strategy to do the enumeration, and use *selectedSet* and *candidateSet* to store those objects already selected and the candidate objects that could contribute to the best group containing the pivot with objects in *selectedSet*.

Since the first part of the cost function of the MIN+MAX SGK query is determined by the nearest object to q in a group, we first choose an object o_n to be the nearest object to q and put it into *selectedSet*. Then, in the following search, an object o_j should not be taken into consideration if $\mathsf{Dist}(o_j, q) < \mathsf{Dist}(o_n, q)$. We process the objects in ascending order of their distances to q. The text pruning and the distance pruning strategies are still applicable, and the termination condition is also the same as that in MAXMAX-Exact.

The exact algorithm for the MIN+MAX SGK query is similar to Algorithm 5. We use the IR-tree to find those objects containing the most infrequent keyword t_{inf} and, when the distance to q exceeds $curCost$, we terminate the algorithm. After we get an object containing t_{inf}, we first obtain its search range according to Lemma 6.3 utilizing the IR-tree. Then we do exhaustive search enhanced with the pruning strategies to find the best group containing the pivot. After each pivot is processed, we return the group with the smallest cost.

Assume that there are n query keywords, the number of relevant objects is m, and the number of objects in the search range around an object is r in the worst case. Similar to MAXMAX-Exact, the complexity of this algorithm is also $O(\frac{m}{n}r^n)$.

7. PROCESSING TOP-*K* SPATIAL GROUP KEYWORD QUERIES

The proposed algorithms for answering the SGK queries can easily be extended to answer the corresponding top-k SGK (kSGK) queries. We cover the exact top-k algorithms for the SUM, MAX+MAX, and MIN+MAX SGK queries in Section 7.1, Section 7.2, and Section 7.3, respectively.

7.1. Top-*k* SUM SGK Query

We extend the exact algorithm presented in Section 4.2 to efficiently answer the SUM kSGK query. We still utilize the IR-tree, and we process the objects in ascending order of their distances to a given query q. To find the top-k results, the stopping condition is changed to these two: (1) we reach the k-th object covering all the query keywords; and (2) we reach an object such that its distance to q is larger than that of the current k-th group covering all the query keywords.

The idea of this algorithm is as follows. We use an array of priority queues to store the top-k groups for each query keyword subset. Each group is represented by a 3-tuple ($cost$, $objects$, $status$). We utilize the IR-tree to process objects that contain some query keywords according to ascending order of their distances to the query, as we do in Algorithm 2.

In the SUM-ExactWIndex algorithm, each query keyword subset has a status telling whether its lowest cost has been found. This can be used to prune objects and MBRs that cover a subset that already has the lowest cost. We need to do similarly for finding the top-k results. Since each keyword subset has k groups, we also need to keep track of the status for each of them, that is, whether they are really in the top-k list of this keyword subset. When we reach an object e, those groups with cost values smaller than the distance of e to q change their status to "marked." When all the top-k groups of a keyword subset have status "marked," the status of this subset is changed to "marked," and it is further used to prune objects and MBRs.

The object e is then used to update the top-k groups of each keyword subset that is a subset of $e.\psi \cap q.\psi$. If this object is the k-th group of a keyword subset, we know that all the top-k groups have been found for this subset, and we move it to $markedSet$. Finally, this object e is used to generate new groups with existing groups, as done in lines 35–44 in SUM-ExactWIndex. We enumerate each query keyword subset and, for each group of this subset, it is combined with e to generate a new group for the keyword subset that is the union of $e.\psi$ and the current keyword subset. When the termination condition is satisfied, we stop the algorithm and return the k groups obtained.

In this algorithm, we maintain k groups for each keyword subset and, when we generate a new group for a keyword subset, we need to compare it with the existing groups of this subset, which has time complexity $O(\log k)$. Thus the complexity of this algorithm is $O(k \log k)$ times that of algorithm SUM-ExactWIndex.

7.2. Top-*k* MAX+MAX SGK Query

It is straightforward to extend the MAXMAX-Exact algorithm to answer the MAX+MAX kSGK query. Recall that in MAXMAX-Exact, we first invoke MAXMAX-Appro2 to obtain a feasible group, the cost of which serves as an upper bound on the optimal cost. This upper-bound cost can be utilized to obtain an exhaustive search range that is updated during the search. Thus we just need to first obtain k groups approximately, and then we can do the same exhaustive search as in MAXMAX-Exact to find the k best groups.

We extend the MAXMAX-Appro1 algorithm to approximately find k feasible groups. First, we invoke MAXMAX-Appro1 to find a feasible group G_1. Then, all the objects in G_1 are inserted into a queue in ascending order of their distances to q. We find the nearest object o_n to q from the queue and, for each keyword in $o_n.\psi$, we find the next

nearest object containing this word, and replace o_n with these objects to obtain a new group G_2. The objects in G_2 not accessed yet are also inserted into the queue according to their distances to q. We similarly process all the remaining objects in the queue. After selecting an object from the queue, we find all the discovered groups containing it, and we replace this object with further objects covering the same keyword subset in those groups to generate new groups. We stop this procedure when k groups are found.

After we find k feasible groups, the largest cost of these groups is the upper bound of the cost of the exact k-th group, and we denote this value by $curCost_k$. Then we find the most infrequent query keyword t_{inf}, and we process the objects containing t_{inf} (the pivot) in ascending order of their distances to q. For each pivot, we obtain a search space utilizing Lemma 5.9 by replacing $curCost$ with $curCost_k$, and we find the best group containing the pivot by invoking Function enumerateBestGroup. If we find a group whose cost is smaller than $curCost_k$, we update the top-k list of groups and the value of $curCost_k$. When we reach an object whose distance is even larger than $curCost_k$, we can stop and return the k groups already found. We denote this algorithm as TOPK-MAXMAX-Exact.

7.3. Top-k MIN+MAX SGK Query

Similar to the exact algorithm for processing the MAX+MAX kSGK query, we extend the MINMAX-Exact algorithm to answer the MIN+MAX kSGK query. We still use the method introduced in Section 7.2 to approximately find k groups, and we initialize the current best k-th group's cost, $curCost_k$, as the largest cost of these groups. Then we find the most infrequent query keyword t_{inf}, and utilize Lemma 6.3 to obtain the search space around a pivot, that is, an object containing t_{inf}. We process the pivots in ascending order of their distances to q until we reach a distance to q that exceeds $curCost_k$. We denote this algorithm as TOPK-MINMAX-Exact.

8. WEIGHTED SPATIAL GROUP KEYWORD QUERY

We present the processing of the weighted SUM, MAX+MAX, and MIN+MAX SGK queries in Sections 8.1, 8.2, and 8.3, respectively.

8.1. Processing the Weighted SUM SGK Query

The cost function of the weighted SUM SGK query is defined as $\mathsf{Cost}(q, \chi) = \sum_{o \in \chi} \mathsf{wDist}(o, q)$, where the inter-object relationship is not considered. Hence this query can be easily answered by modifying the approximation and exact algorithms for the SUM SGK query. We only need to replace the original Euclidean distance by the weighted distance in the algorithms as introduced in Sections 4.1 and 4.2.

The following lemma makes it possible to estimate the lower bound of the weight of an object in a given IR-tree node R, given a query q. We define $\mathsf{Wt}(R, q) = e^{-\mathsf{Sim}(R,q)}$, where $\mathsf{Sim}(R, q)$ computes the relevance between q and the pseudo document of R.

LEMMA 8.1. *Given a query q and an IR-tree node R, $\forall\, o \in R\ (\mathsf{Wt}(o, q) \geq \mathsf{Wt}(R, q))$.*

PROOF. Recall that in $R.\psi$, the weight of each term is the maximum weight of t in any document contained in the subtree rooted at node R. Hence, according to Eq. (2), it is true that $\forall\, o \in R\ (\mathsf{Sim}(o, q) \leq \mathsf{Sim}(R, q))$ and thus $\mathsf{Wt}(o, q) \geq \mathsf{Wt}(R, q)$. □

With Lemma 8.1, we are able to compute the lower bound of the weighted distance of any object o in a given node R to q as $\mathsf{Wt}(R, q) \cdot \mathsf{minDist}(R, q)$ since $\mathsf{minDist}(R, q)$ is the minimum distance from q to R.

In order to answer the weighted SUM SGK query approximately, we only need to modify Algorithm 1 as follows: in line 19, we compute $dist$ as $\mathsf{Wt}(p', q) \cdot \mathsf{minDist}(p', q)$; and in line 20, we compute $dist$ as $\mathsf{wDist}(p', q)$. The approximation ratio of this algorithm

remains $\sum_{i=1}^{n} \frac{1}{i}$, where n is the number of query keywords. The complexity of the algorithm also does not change.

The exact algorithm can be obtained by modifying Algorithm 2 as follows: in line 12, we enqueue a node p' into the queue using key $\mathsf{Wt}(p', q) \cdot \mathsf{minDist}(P', q)$; and we replace all $\mathsf{Dist}(o, q)$ by $\mathsf{wDist}(o, q)$ in lines 15, 19, 30, and 41. The complexity of the algorithm is unaffected.

When the weight is computed as a user rating or popularity score which is dependent on the query, it is easier to compute $\mathsf{Wt}(R, q)$. We just need to store the minimum value of $\mathsf{Wt}(o, q)$ in each node R in the IR-tree when building the index.

8.2. Processing the Weighted MAX+MAX SGK Query

8.2.1. Approximation Algorithm. Recall that in Algorithm 3 we find the nearest object for each keyword in query q, and we then use the group of these objects to answer the unweighted query. In the weighted MAX+MAX query, this method is not appropriate since the nearest object may have a low weight. Hence, we propose to find instead that object for each query keyword with the smallest weighted distance, that is, for each keyword $t \in q.\psi$, we find object $o_t = \arg\min_{t \in o_t.\psi}(\mathsf{wDist}(o_t, q))$, and we return the group $G_{app} = \bigcup_{t \in q.\psi} o_t$. We call this algorithm WMAXMAX-Appro. The following theorem gives an approximation ratio for the algorithm.

THEOREM 8.2. *The approximation ratio of WMAXMAX-Appro is 1+2e.*

PROOF. Let o_f be the furthest-away object, let o_w be the object that has the largest weighted distance to q in G_{app}, and let $d = \mathsf{wDist}(o_w, q)$. Denoting the optimal group by G_{opt}, we have $\mathsf{Cost}(G_{opt}) \geq d$ since there must exist an object in G_{opt} containing the same query keyword as does o_w. The largest diameter of G_{app} is $2\mathsf{Dist}(o_f, q)$. Since $\mathsf{Wt}(o_f, q) \cdot \mathsf{Dist}(o_f, q) \leq d$, $\mathsf{Dist}(o_f, q) \leq d/\mathsf{Wt}(o_f, q)$. Hence $\mathsf{Cost}(G_{app}) \leq d + \frac{\max_{o \in G_{app}} \mathsf{Wt(o,q)}}{\mathsf{Wt(o_f,q)}} 2d$. Because $\max_{o \in G_{app}} \mathsf{Wt}(o, q) \leq 1$ and $\mathsf{Wt}(o_f, q) \geq e^{-1}$, we obtain $\mathsf{Cost}(G_{app}) \leq (1 + 2e)\mathsf{Cost}(G_{opt})$. □

If the object weights represent other keyword-independent attributes such as user ratings, the ratio for the algorithm is $1 + 2e^{w_{max} - w_{min}}$, where w_{max} is the maximum weight and w_{min} the minimum weight. This is because $\max_{o \in G_{app}} \mathsf{Wt}(o, q) \leq e^{-w_{min}}$ and $\mathsf{Wt}(o_f, q) \geq e^{-w_{max}}$ and, similar to the proof of Theorem 8.2, we obtain $\mathsf{Cost}(G_{app}) \leq (1 + 2e^{w_{max} - w_{min}})\mathsf{Cost}(G_{opt})$.

8.2.2. Exact Algorithm. The MAX+MAX query is the special case of the weighted MAX+MAX query where all objects have the same weight, and thus it is more challenging to develop an exact algorithm for the weighted MAX+MAX query.

To compute the weighted query, we first invoke WMAXMAX-Appro to derive a feasible group of objects. We then calculate the cost of this group, which is an upper bound on the cost of the optimal group for a given query. Then we follow the idea of MAXMAX-Exact to process those objects containing the most infrequent query keyword t_{inf} in ascending order of their weighted distance. Without loss of generality, we still call such an object a *pivot*. We bound the space to search around each pivot by checking whether an object o and the pivot can together form a group that has a cost smaller than the current best cost. Finally, we perform an exhaustive search with the following pruning strategies in each search space, and update the current best cost correspondingly.

I. Bounding the exhaustive space around an object containing t_{inf}. We can reduce the exhaustive search around an object containing t_{inf} in a similar way as for MAXMAX-Exact.

LEMMA 8.3. *Given a weighted* MAX+MAX *SGK query q and two objects o_i and o_j, a lower-bound value of the cost of a group G containing o_i and o_j can be computed by* $\max(\mathsf{wDist}(o_i, q), \mathsf{wDist}(o_j, q)) + \max(\mathsf{Wt}(o_i, q), \mathsf{Wt}(o_j, q)) \cdot \mathsf{Dist}(o_i, o_j)$. *We call this value $LOW_q(o_i, o_j)$ (with a slight abuse of notation).*

PROOF. First, we have $\max(\mathsf{wDist}(o_i, q), \mathsf{wDist}(o_j, q)) \leq \max_{o \in G}(\mathsf{wDist}(o, q))$. It is also true that $\max(\mathsf{Wt}(o_i, q), \mathsf{Wt}(o_j, q)) \cdot \mathsf{Dist}(o_i, o_j) \leq \max_{o \in G} \mathsf{Wt}(o, q) \cdot \max_{o_1, o_2 \in G} \mathsf{Dist}(o_1, o_2)$, and thus $LOW_q(o_i, o_j) \leq \mathsf{Cost}(q, G)$. □

If $LOW_q(o_i, o_j)$ exceeds *curCost* (the current best cost, initialized as the cost obtained by WMAXMAX-Appro), we know that o_i and o_j cannot contribute to the optimal group together. Since the distance is no longer the only criterion when computing the cost, we cannot determine a region as in Lemma 5.9 because the weight of the object is unknown. However, given a pivot o, we can find all candidate objects that can be combined with it to form the optimal group using Lemma 8.3. That is, the search space around a pivot o is $\{o_x | LOW_q(o, o_x) < curCost\}$.

II. Enumerating the best group in the search space. We proceed to explain how to exhaustively search around a pivot. The procedure is similar to that of the MAXMAX-Exact algorithm. We adopt the depth-first search strategy to do the enumeration, and we maintain two sets *selectedSet* and *candidateSet* to store the selected objects and the candidate objects that can possibly contribute to the optimal group containing the pivot with objects in *selectedSet*.

We enumerate the objects in ascending order of their weighted distances to the query. When processing an object o_m, we assume it is the object that has the largest weighted distance in the group and put it in *selectedSet*. Then, in the subsequent search, we only consider objects whose weighted distance is smaller than $\mathsf{wDist}(o_m, q)$. The text pruning and the termination condition are the same as in MAXMAX-Exact. In the distance pruning strategy, we need to compute the cost of *selectedSet* under the new cost function to check whether it exceeds *curCost*.

The algorithm can be obtained by modifying Algorithm 5 as follows: (1) We order the priority queue by the weighted distance instead of the original distance to q. In line 12, we replace $\mathsf{Dist}(p', q)$ by $\mathsf{wDist}(p', q)$; and in line 13, we replace $\mathsf{minDist}(p', q)$ with $\mathsf{Wt}(p', q) \cdot \mathsf{minDist}(p', q)$; (2) In line 24, we compute $LOW_q(p, p')$ as in Lemma 8.3 rather than as in Lemma 5.8.

8.3. Processing the Weighted MIN+MAX SGK Query

8.3.1. Approximation Algorithm. The algorithm WMAXMAX-Appro can be used to find a group for a given weighted MIN+MAX query as well. We show that this algorithm also has a performance bound when used to answer the weighted MIN+MAX query.

THEOREM 8.4. *Given a weighted* MIN+MAX *query q, the cost of the group G_{app} returned by WMAXMAX-Appro is at most $3e^2$ times the cost of the optimal group G_{opt}.*

PROOF. Let o_f be the furthest-away object in G_{app}, and o_w be the object in G_{app} that has the smallest weighted distance to q. Hence $\mathsf{Cost}(G_{app}, q) \leq \mathsf{wDist}(o_w, q) + 2\max_{o \in G_{app}}(\mathsf{Wt}(o, q)) \cdot \mathsf{Dist}(o_f, q)$. Since o_w has the smallest weighted distance, that is, $\mathsf{wDist}(o_w, q) \leq \mathsf{wDist}(o_f, q)$, $\mathsf{Cost}(G_{app}, q) \leq \mathsf{wDist}(o_f, q) + 2\max_{o \in G_{app}}(\mathsf{Wt}(o, q)) \cdot \mathsf{Dist}(o_f, q) \leq 3\max_{o \in G_{app}}(\mathsf{Wt}(o, q)) \cdot \mathsf{Dist}(o_f, q)$.

Let o_t be the object that contains the same query keyword as does o_f in G_{opt}, and o_n be the object that has the smallest weighted distance in G_{opt}. Hence $\mathsf{Cost}(G_{opt}, q) \geq \mathsf{wDist}(o_n, q) + \max_{o \in G_{opt}}(\mathsf{Wt}(o, q)) \cdot \mathsf{Dist}(o_n, o_t) \geq \mathsf{Wt}(o_n, q) \cdot (\mathsf{Dist}(o_n, q) + \mathsf{Dist}(o_n, o_t)) \geq \mathsf{Wt}(o_n, q) \cdot \mathsf{Dist}(o_t, q)$.

According to the algorithm, $\mathsf{wDist}(o_f, q) \leq \mathsf{wDist}(o_t, q)$, and we get $\frac{\mathsf{Dist}(o_f,q)}{\mathsf{Dist}(o_t,q)} \leq \frac{\mathsf{Wt}(o_t,q)}{\mathsf{Wt}(o_f,q)} \leq e$. Finally, we obtain $\frac{\mathsf{Cost}(G_{app},q)}{\mathsf{Cost}(G_{opt},q)} \leq \frac{3\max_{o\in G_{app}} \mathsf{Wt}(o,q)\cdot\mathsf{Dist}(o_f,q)}{\mathsf{Wt}(o_n,q)\cdot\mathsf{Dist}(o_t,q)} \leq 3e^2$. □

Generally, let w_{max} be the maximum weight and w_{min} be the minimum weight, we obtain the ratio $3e^{2(w_{max}-w_{min})}$ for this algorithm. This is because $\frac{\mathsf{Cost}(G_{app},q)}{\mathsf{Cost}(G_{opt},q)} \leq e^{(w_{max}-w_{min})}$ and $\frac{\max_{o\in G_{app}} \mathsf{Wt}(o,q)}{\mathsf{Wt}(o_n,q)} \leq e^{(w_{max}-w_{min})}$.

8.3.2. Exact Algorithm. We first utilize the WMAXMAX-Appro algorithm to derive a feasible group of objects. We then calculate the cost using the weighted MIN+MAX cost function. The cost is then an upper bound on the cost of the optimal group of a given query q. Then we follow the idea of MINMAX-Exact to process those objects containing the most infrequent query keyword t_{inf} in ascending order of their weighted distances to the query. We bound the search space around each pivot. That is, we find all objects that are possible to combined with the pivot to form a group with a cost that is smaller than that of the current best group. Finally, we perform an exhaustive search with the pruning strategies around each pivot and accordingly update the current best cost.

I. Bounding the exhaustive space space around an object containing t_{inf}. Recall that, in the MINMAX-Exact algorithm, we utilize Lemma 6.3 to obtain the search space around a pivot. However, distance is not the only factor when computing the cost of a group in a weighted query. The following lemma can be used to bound the search space around a pivot.

LEMMA 8.5. *Given a weighted* MIN+MAX *query q and an object o_i containing t_{inf}, the most infrequent keyword in $q.\psi$, an object o_j can be pruned in the search space around o_i if* $\max(\mathsf{Wt}(o_i, q), \mathsf{Wt}(o_j, q)) \cdot \mathsf{Dist}(o_i, o_j) > curCost$, *where $curCost$ is the current best cost.*

PROOF. It is obvious that $\max(\mathsf{Wt}(o_i, q), \mathsf{Wt}(o_j, q)) \cdot \mathsf{Dist}(o_i, o_j) \leq \max_{o\in G} \mathsf{Wt}(o, q) \cdot \max_{o_1,o_2\in G} \mathsf{Dist}(o_1, o_2)$. Thus we can never obtain a group with cost less than $curCost$ if both of o_i and o_j are selected. □

II. Enumerating the best group in the search space. The procedure is similar to that of the MINMAX-Exact algorithm. We adopt the depth-firstsearch strategy to do the enumeration, and we maintain two sets, *selectedSet* and *candidateSet*, to store the selected objects and candidate objects that may contribute to the optimal group containing the pivot with objects in *selectedSet*. We then enumerate the objects in ascending order of their weighted distances to the query. We first choose an object o_m to be the object with the minimum weighted distance to q. In subsequent search, only objects with weighted distance smaller than $\mathsf{wDist}(o_m, q)$ are considered. The pruning strategies remain applicable. The only difference is that we do distance pruning using the weighted MIN+MAX cost function.

9. EXPERIMENTAL STUDY

9.1. Experimental Settings

Algorithms. For the SUM spatial group keyword query, we consider the approximation algorithm from Section 4.1 (denoted by SUM-A for short), and the exact algorithm utilizing the IR-tree from Section 4.2 (denoted by SUM-E). We also show the results of the exact algorithm when using the IR-tree [Cao et al. 2011], which is denoted by SUM-EP. For the MAX+MAX spatial group keyword query, we evaluate the two approximation algorithms from Sections 5.1 and 5.2 (denoted by MAXM-A1 and MAXM-A2, respectively) and the exact algorithm from Section 5.3 (denoted by MAXM-E). We also evaluate the approximation algorithm and the exact algorithm

Table VII. Dataset Properties

Property	Web	GN	Hotel
Number of objects	579,727	1,868,821	20,790
Number of unique words	2,899,175	222,409	602
Number of words	249,132,883	18,374,228	80,845

proposed in the previous work [Cao et al. 2011], denoted by MAXM-AP and MAXM-EP, respectively. We compare our algorithms with the approximation algorithm and the exact algorithm proposed in the work Long et al. [2013] as well, denoted by LONG-A and LONG-E, respectively. For the MIN+MAX spatial group keyword query, we study the approximation algorithm from Section 6.1 (denoted by MINM-A) and the exact algorithm from Section 6.2 (denoted by MINM-E).

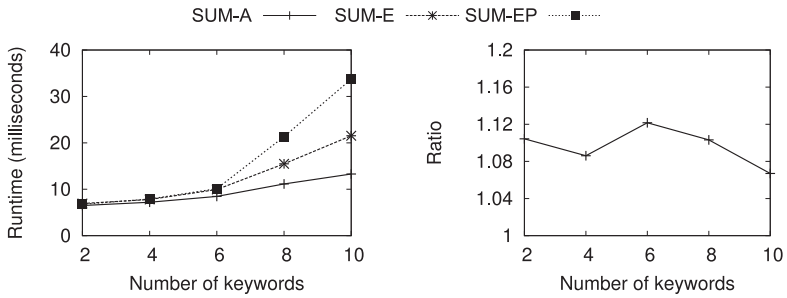
We also conduct experiments with the algorithms for kSGK queries, that is, the exact algorithm of the top-k SUM SGK query from Section 7.1 (denoted by K-SUM-E for short), the exact algorithm (denoted by K-MAXM-E) of the top-k MAX+MAX SGK query from Section 7.2, and the exact algorithm (denoted by K-MINM-E) of the top-k MIN+MAX SGK query from Section 7.3. The approximation and exact algorithms for the weighted SUM, MAX+MAX, and MIN+MAX SGK queries are also evaluated, denoted by WSUM-A, WSUM-E, WMAXM-A, WMAXM-E, WMINM-A, and WMINM-E, respectively.

Dataset and queries. We use three datasets. Table VII shows some properties of these datasets. Dataset GN is extracted from the U.S. Board on Geographic Names (geonames.usgs.gov). Here, each object is a location with a geographic name (e.g., valley). Dataset Web is generated from two real datasets. The first is WEBSPAM-UK2007[2] that consists of a large number of Web documents; the second is a spatial dataset containing the tiger Census blocks in Iowa, Kansas, Missouri, and Nebraska (www.rtreeportal.org). We randomly combine Web documents and spatial objects to get the Web dataset. Dataset Hotel contains spatial objects that represent hotels in the U.S. (www.allstays.com). Each object has a location and a set of words that describe the hotel (e.g., restaurant, pool). Hotel is small and is used to evaluate the performance of our algorithms when the dataset and index are memory resident, and the other two large datasets are used to evaluate our algorithms when the dataset and index are disk based.

We generate five query sets in the space of GN, in which the number of keywords is 2, 4, 6, 8, and 10, respectively. We also generate five similar query sets in the space of both Web and Hotel. Each set comprises 50 queries. When generating a query set with n keywords, we randomly select n objects from the dataset and then select a keyword from each object as a query keyword, and the centerpoint of these objects is used as the query location. Such queries would need similar processing time, and we report the average cost of queries in each query set. We also conduct experiments on queries generated in other ways, and a summary of the experimental results is presented in Section 9.2.4.

Setup. The IR-tree index structure is disk resident, and the page size is 4KB. The number of children of a node in the IR-tree is computed given the fact that each node occupies a page. This translates to 100 children per node in our implementation. All algorithms were implemented in C++ and run in Windows 7 System on an Intel(R) Xeon(R) CPU E5-1620 @2.66 GHz with 8GB RAM.

[2]http://barcelona.research.yahoo.net/webspam/datasets/uk2007.



(a) runtime (b) approximation ratio

Fig. 5. Results of SUM SGK queries on Hotel.

9.2. Experimental Results on SGK Queries

9.2.1. The SUM *SGK Query.*

I. Results on Hotel. This experiment studies the performance of our algorithms when the dataset and index are in memory. Specifically, we study the efficiency and accuracy of the four algorithms when we vary the number of query keywords on Hotel.

Figure 5(a), where the y-axis is in logarithmic scale, shows the runtime of the five algorithms on Hotel. As expected, the approximation algorithm SUM-A runs faster than all the exact algorithms, that is, SUM-E and SUM-EP. The runtime of the approximation algorithm SUM-A increases almost linearly with the number of query keywords. It is understandable that its running time is proportional to the number of query keywords: SUM-A keeps searching for the object with the lowest cost that covers part or all of the query keywords, and it terminates when a group of objects that covers the query keywords has been found.

For the exact algorithms, SUM-E avoids scanning objects that do not contain query keywords by utilizing the IR-tree and can avoid accessing those objects whose distances to the query are larger than the cost of the discovered group, thus significantly pruning the search space. The experimental results demonstrate the usefulness of the IR-tree-based pruning strategies. SUM-EP also utilizes the IR-tree. However, as analyzed in Section 4.2, when we reach an object e, all the keyword subsets of $e.\psi$ are considered in SUM-EP, which takes more time. Hence it runs slower than SUM-E.

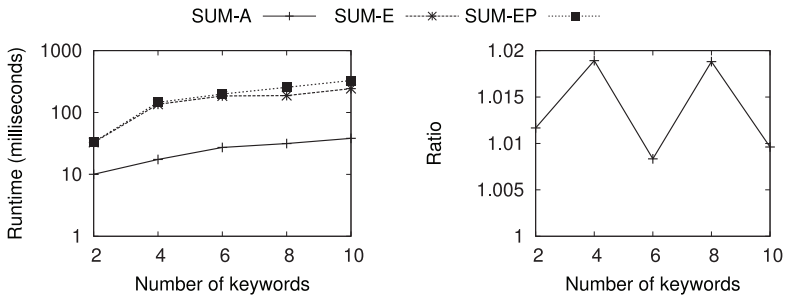
It can also be observed that the runtime of both exact algorithms increases with the number of query keywords; however, the increase is not exponential. The reason is that computing the costs of objects dominates the running time over the dynamic programming component.

Figure 5(b) shows the accuracy of SUM-A on Hotel. It shows that the approximation algorithm is capable of achieving very accurate results.

We also conduct experiments on Hotel when the data and index are disk based, and we observe qualitatively similar results.

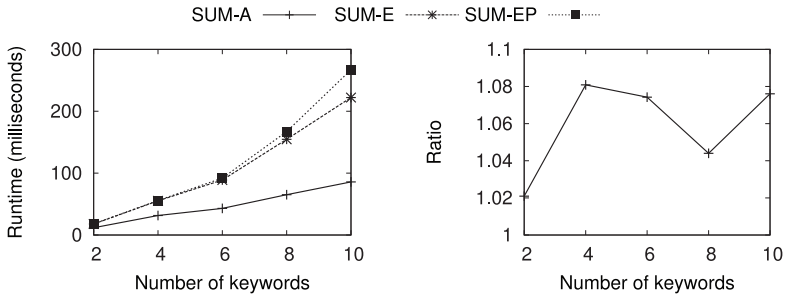
II. Results on GN. The objective of this set of experiments is to study the efficiency and accuracy of the four algorithms when we vary the number of query keywords on GN. Figure 6(a) shows the runtime of the algorithms on the dataset GN, and Figure 6(b) shows the accuracy of SUM-A on GN.

Since this dataset contains a large amount of objects, the gap between the running time of the approximation algorithm and the exact algorithms is quite obvious. We can see the approximation algorithm SUM-A runs much faster than the exact algorithms. The runtime of the approximation algorithm SUM-A increases almost linearly with



(a) runtime (b) approximation ratio

Fig. 6. Results of SUM SGK queries on GN.



(a) runtime (b) approximation Ratio

Fig. 7. Results of SUM SGK queries on Web.

the number of query keywords. We can also see that SUM-E still outperforms SUM-EP over all five query sets.

The I/O cost is consistent with the runtime and thus not reported.

III. Results on Web. This experiment studies the efficiency and accuracy on the dataset Web in which each object is associated with a large set of keywords. Figure 7(a) shows the runtime of the algorithms SUM-A, SUM-E, and SUM-EP, and Figure 7(b) shows the accuracy of SUM-A. We observe qualitatively similar results on Web as on GN.

9.2.2. The MAX+MAX *SGK Query.* We first evaluate the performance of our algorithms on the three datasets, and then compare our approximation and exact algorithms with the algorithms proposed by Long et al. [2013].

I. Results on Hotel. This experiment studies the performance of our algorithms when the dataset and index are in memory. Specifically, we study the efficiency and accuracy of our three approximation algorithms (i.e., MAXM-A1, MAXM-A2, and MAXM-AP) and two exact algorithms (i.e., MAXM-E1 and MAXM-EP) when we vary the number of query keywords on Hotel.

Figure 8(a) shows the runtime of these algorithms, and Figure 8(b) shows the accuracy of the approximation algorithms. Note that, due to the hardness of answering the MAX+MAX SGK query, the exact algorithm may spend too much time on finding the optimal group (for some queries, the two algorithms cannot return an answer within one day). Therefore, to ensure the readability of the figures, we set a timeout threshold to 5 minutes. If the exact algorithm fails to find a group within this threshold for a query, we terminate the algorithm. Figure 8(c) shows the success rate of the three

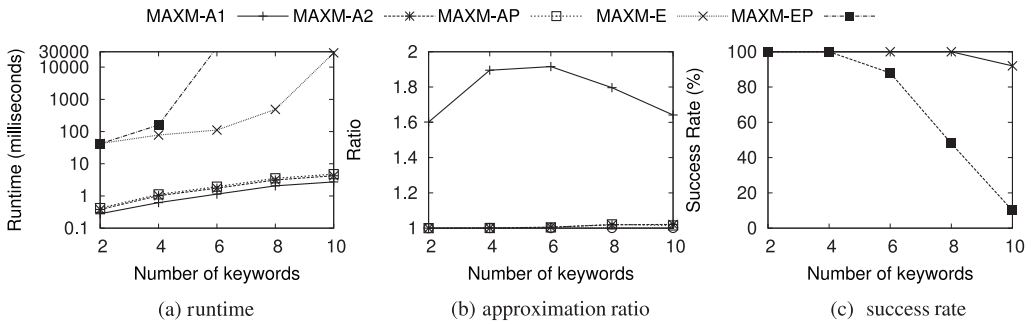


Fig. 8. Results of MAX+MAX SGK queries on Hotel.

exact algorithms, that is, the percentage of queries on which the two algorithms can return an answer within the predefined timeout threshold.

As can be seen, MAXM-A1 outperforms MAXM-A2 and MAXM-AP in terms of runtime, and the accuracy of MAXM-A1 is worse than those of MAXM-A2 and MAXM-AP. This is because MAXM-A1 terminates once a group of nearest objects covering each query keyword is found. In contrast, MAXM-A2 and MAXM-AP invoke MAXM-A1 multiple times. MAXM-A2 runs slightly faster than does MAXM-AP, but this small difference is not visible in the figure. The reason for the difference is that we only invoke MAXM-A1 on those objects containing the most infrequent query keywords. MAXM-A1 usually needs fewer invocations than MAXM-AP. Both MAXM-A2 and MAXM-AP achieve good accuracy compared with the optimal group returned by MAXM-E.

MAXM-E and MAXM-EP are able to find the optimal group. Due to their time complexity, they are much slower than the approximation algorithms. As expected, with an increase in the number of keywords, the runtime of MAXM-E increases exponentially due to its exhaustive search around pivot objects. MAXM-EP performs much worse than MAXM-E because it spends too much time on enumerating the possible combinations of IR-tree nodes. As shown in Figure 8(c), the success rate of MAXM-EP drops dramatically as the number of query keywords increases. It is able to give an answer only on 10% of the queries (five queries) when we set the number of query keywords to 10. MAXM-E can answer almost all the queries, and only fails on three queries containing 10 keywords.

When the number of keywords is small (e.g., no larger than 10), the runtime of the exact algorithms is reasonable for applications without a high demand on the query time, for instance, finding research partners. However, the approximation algorithms represent a better option when query time is essential.

II. Results on GN. Figure 9(a) shows the runtime of MAXM-A1, MAXM-A2, MAXM-AP, MAXM-E, and MAXM-EP, Figure 9(b) shows the accuracy of MAXM-A1, MAXM-A2, and MAXM-AP, and Figure 9(c) shows the success rate of MAXM-E and MAXM-EP. Because GN contains a large number of objects, the number of candidate objects is usually large and thus the query time is much longer than that on Hotel. The success rate of MAXM-E drops quickly as the number of query keywords increases. MAXM-EP can only find the answer for one query on the query set containing eight keywords, and fails on all queries when the number of query keywords is 10.

We also notice that, when the query contains very few keywords, MAXM-AP may perform better than MAXM-A2. The reason may be that, in MAXM-A2, we only invoke MAXM-A1 on objects within a certain range around the query point containing the most infrequent query keyword. However, the keyword frequency is computed based

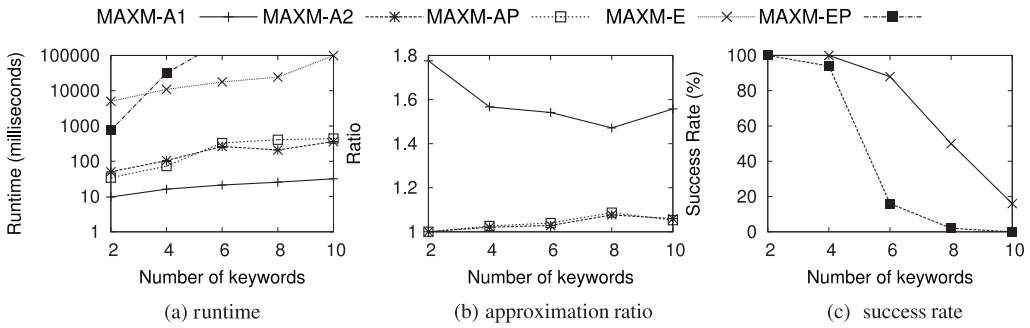


Fig. 9. Results of MAX+MAX SGK queries on GN.

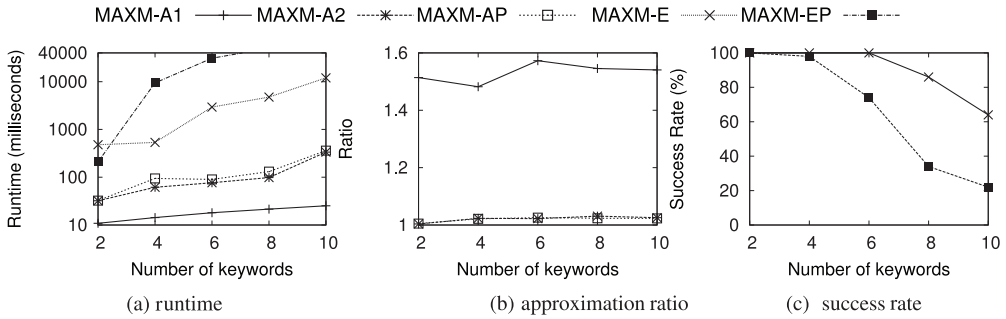


Fig. 10. Results of MAX+MAX SGK queries on Web.

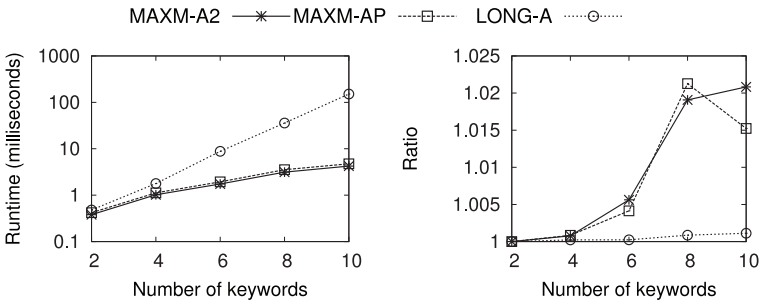
on the whole dataset, and thus the "globally" most infrequent keyword may not be the most infrequent keyword within the candidate range around the query point.

III. Results on Web. Figure 10(a) shows the runtime of MAXM-A1, MAXM-A2, MAXM-AP, MAXM-E, and MAXM-EP, Figure 10(b) shows the accuracy of MAXM-A1, MAXM-A2, and MAXM-AP, and Figure 10(c) shows the success rate of MAXM-E and MAXM-EP. We observe qualitatively similar results on Web as we do on Hotel.

IV. Comparison with Existing Work. Long et al. [2013] study the problem of answering MAX+MAX SGK queries. We first briefly review their approximation algorithm (denoted by LONG-A) and their exact algorithm (denoted by LONG-E) and then we compare our algorithms with these two.

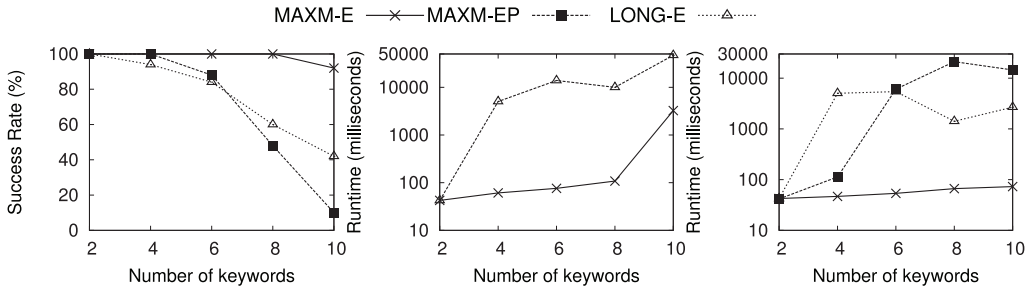
LONG-A processes the objects containing at least one query keyword in ascending order of their distances to query q. On each such object o, it invokes MAXM-A1 to find a group for query $\langle o.\lambda, q.\psi \setminus o.\psi \rangle$, and then it computes the cost of this group with respect to the original query q. If the group is better than the current best group, the current best cost is updated. It terminates when reaching an object whose distance exceeds the current best cost. LONG-A has an approximation ratio 1.375.

LONG-E uses the concept of "query distance owner" of a group, which is the furthest-away object to the query in the group, and the "pairwise distance owners" of a group, which is the pair of two objects that have the largest distance. Then, the query distance owner and the pairwise distance owners are enumerated to find the optimal group. The objects are processed in ascending order of their distances to the query during the enumeration. Each object o is selected as the query distance owner. Then, each pair of objects (o_1, o_2) (o_1 and o_2 are closer to q than is o) is selected as the pairwise distance owners. Next, groups with o as the furthest-away object and (o_1, o_2) as the pair contributing the largest diameter are enumerated. If a better group



(a) runtime (b) approximation ratio

Fig. 11. Comparison of approximation algorithms on Hotel.



(a) success rate (b) runtime (c) runtime

Fig. 12. Comparison of exact algorithms on Hotel.

is found, the current best cost is updated. This process is repeated until reaching an object whose distance exceeds the current best cost.

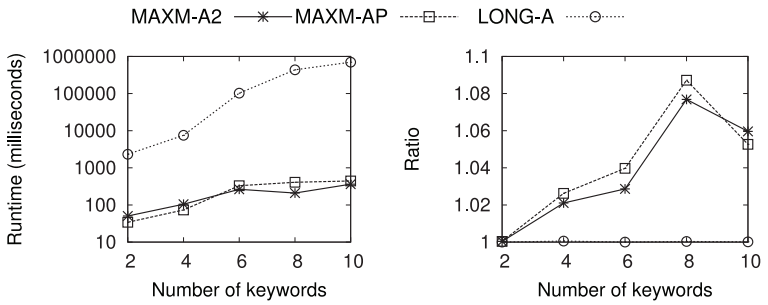
We first compare our algorithms with LONG-A and LONG-E on Hotel, where the dataset and index are kept in memory[3].

Figures 11(a) and 11(b) show the comparison of our approximation algorithms MAXM-A2 and MAXM-AP with LONG-A in terms of efficiency and accuracy, respectively. Because MAXM-A2, MAXM-AP, and LONG-A all outperform MAXM-A1 significantly with respect to accuracy, we do not show the results of MAXM-A1 here. It can be observed that LONG-A always achieves better accuracy and is able to return nearly optimal results. However, since it needs to invoke MAXM-A1 on all candidate objects containing at least one query keyword, it runs much slower than does MAXM-A2 which only considers objects containing the most infrequent keyword. MAXM-AP first finds the keyword only covered by the furthest-away object in the group found by MAXM-A1, and then it invokes MAXM-A1 on each object containing such keyword; thus it is also much faster than LONG-A.

Figure 12(a) shows the success rate of the exact algorithms MAXM-E, MAXM-EP, and LONG-E. Algorithm MAXM-E is able to successfully return results for most queries, and has a much better success rate than do MAXM-EP and LONG-E. LONG-E can only answer all 50 queries containing two keywords, and fails on three queries among those that contain four keywords.

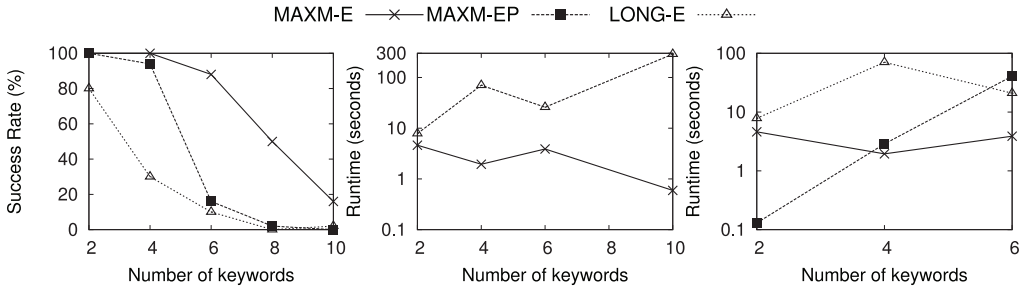
The queries that LONG-E and MAXM-EP can process within the time threshold are always a subset of those that can be answered by MAXM-E. To compare the three

[3]The code implementing LONG-A and LONG-E is provided by the authors of Long et al. [2013].



(a) runtime (b) approximation ratio

Fig. 13. Comparison of approximation algorithms on GN.



(a) success rate (b) runtime (c) runtime

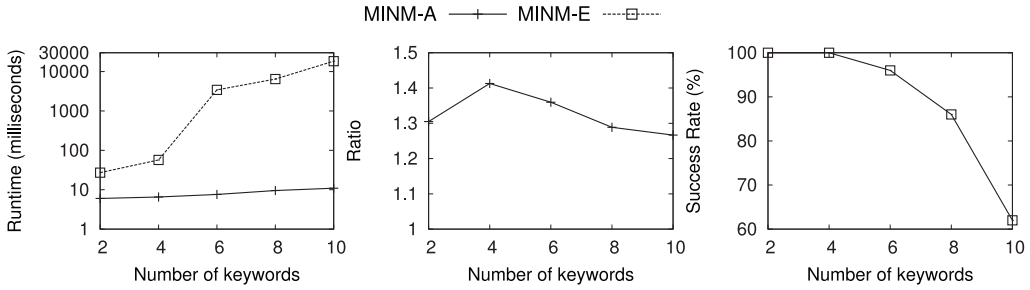
Fig. 14. Comparison of exact algorithms on GN.

exact algorithms, we first run MAXM-E and LONG-E on the queries where LONG-E can successfully return results. The runtime is shown in Figure 12(b). Then, we run the three algorithms on the queries that all of them can answer within the time limit. The runtime, shown in Figure 12(c), indicates that MAXM-E outperforms both MAXM-EP and LONG-E, and that LONG-E performs better than MAXM-EP except on the query set containing four keywords. The reason is that MAXM-EP first does the enumeration in the IR-tree node space and then in the object space, while LONG-E and MAXM-E do the enumeration in the object space directly. In addition, MAXM-E does the enumeration only around those objects containing the most infrequent query keyword, while LONG-E does not consider the term frequency.

We then compare our algorithms with LONG-A and LONG-E on GN, which contains a large amount of objects and calls for a disk-resident index.

Figures 13(a) and 13(b) compare approximation algorithms MAXM-A2 and MAXM-AP with LONG-A in terms of efficiency and accuracy, respectively. The approximation ratio of LONG-A is very close to 1, and it can always obtain nearly optimal groups. However, since it needs to invoke MAXM-A1 on all candidate objects containing at least one query keyword and the index is disk resident, it runs much slower than do MAXM-A2 and MAXM-AP due to the frequent I/O. The ratios of these two algorithms are only slightly worse.

Figure 14(a) shows the success rate of the exact algorithms MAXM-E, MAXM-EP, and LONG-E. LONG-E fails on 10 queries when each query contains only two keywords. It can answer no query containing eight keywords, and can answer only one query containing 10 keywords within the time limit.



(a) runtime (b) approximation ratio (c) success rate

Fig. 15. Results of MIN+MAX SGK queries on Hotel.

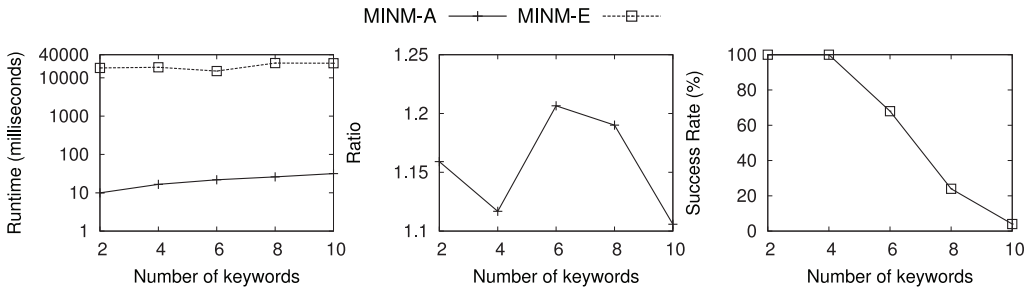
We first run MAXM-E and LONG-E on queries that can be answered by LONG-E. The runtime is shown in Figure 14(b). We then run the three algorithms on queries that can be answered by all of them, reporting the runtime in Figure 14(c). Not all of them can successfully return results on queries containing eight and ten keywords, and we only show the results on queries containing two to six keywords in the figure. It can be observed that MAXM-E consistently outperforms LONG-E. LONG-E performs much worse than does MAXM-EP on queries containing two and four keywords. The reason may be that the pruning strategy in MAXM-EP is efficient on queries with few keywords, and thus the enumeration of many groups is avoided. But as the number of query keywords increases, the enumeration of IR-tree nodes takes longer time, which reduces the efficiency of MAXM-EP.

In conclusion, LONG-A has better or slightly better accuracy, but much longer runtime, than MAXM-A2 and MAXM-AP, especially when the index is disk resident. We note that some queries containing 10 keywords have runtimes that exceed half an hour on GN when LONG-A is used. We also note that the largest accuracy difference between LONG-A and MAXM-A2 is 8% (when the number of keywords is eight), which we believe is low. For the exact algorithms, the success rate of LONG-E is the lowest on queries containing fewer than six keywords. On the queries for which LONG-E succeeds, LONG-E runs faster than does MAXM-EP when queries contain more than four keywords, but it is always much slower than MAXM-E.

9.2.3. The MIN+MAX *SGK Query.*

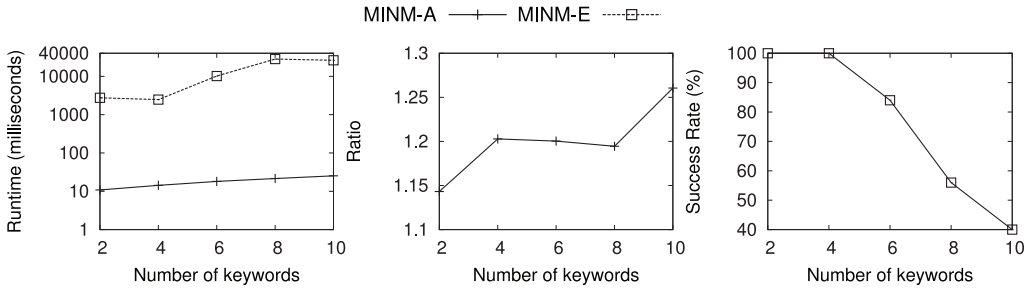
I. Results on Hotel. The objective of this set of experiments is to evaluate the efficiency of the approximation algorithm MINM-A and the exact algorithm MINM-E and to evaluate the accuracy of MINM-A when we vary the number of query keywords. Figure 15(a) shows the runtime of the two algorithms, and Figure 15(b) shows the accuracy of MINM-A on Hotel. Similar to the problem of answering the MAX+MAX SGK query, although we tried to utilize several pruning strategies in the exact algorithm, when too many objects contain the query keywords, MINM-E may fail to give an answer within the timeout threshold. We terminate MINM-E when it fails to find an answer within the time limit, and we report the success rate in Figure 15(c).

As can be seen, MINM-A outperforms MINM-E in terms of runtime. This is because MINM-A terminates once a group of nearest objects covering query keywords is found. MINM-E is much slower. As expected, with an increase in the number of keywords, the runtime of MINM-E increases due to its exhaustive search on pivot objects. It can also be observed that the success rate of the exact algorithm on MIN+MAX SGK queries is much lower than that of the MAX+MAX SGK queries. The reason is that



(a) runtime (b) approximation ratio (c) success rate

Fig. 16. Results of MIN+MAX SGK queries on GN.



(a) runtime (b) approximation ratio (c) success rate

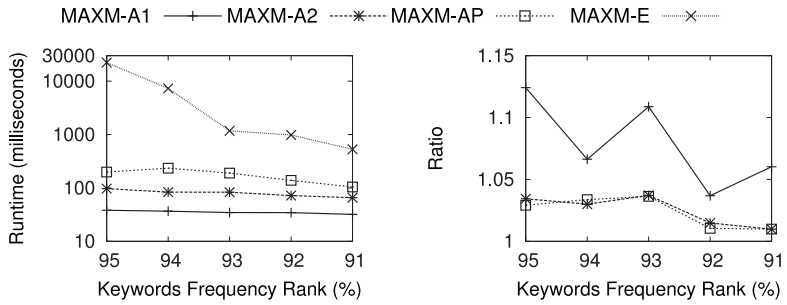
Fig. 17. Results of MIN+MAX SGK queries on Web.

the exhaustive search range of using the MIN+MAX cost function is usually larger than that of using the MAX+MAX cost function.

II. Results on GN. Figure 16(a) shows the runtime of MINM-A and MINM-E, Figure 16(b) shows the accuracy of MINM-A, and Figure 16(c) shows the success rate of MINM-E on GN. We observe qualitatively similar results on GN as we do on Hotel. Since GN contains a large number of objects, the runtime of MINM-E is always much larger than the runtime of MINM-A.

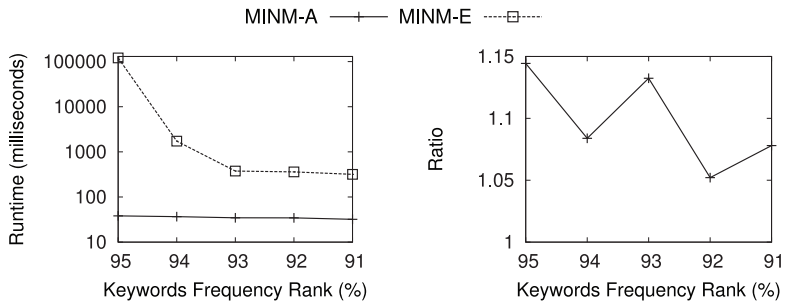
III. Results on Web. Figure 17(a) shows the runtime of MINM-A and MINM-E, Figure 17(b) shows the accuracy of MINM-A, and Figure 17(c) shows the success rate of MINM-E. We observe qualitatively similar results on Web as we do on GN and Hotel.

9.2.4. Effects of Query Keyword Frequency. On the query set generated by randomly selecting the objects and then selecting a keyword from each object, both MAXM-E and MINM-E fail to answer some queries within the 5-minute time limit when the frequency of the query keywords is too high. In this case, there may exist too many objects containing the query keywords that need to be checked during an exhaustive search. We found that, on Web, the number of candidate objects may reach around 30 million for some queries, which exceeds 50% of the dataset! Thus we generate another five query sets containing 10 keywords. In each query set, we still randomly select the objects, but when we select keywords for the query, we omit keywords with very high frequency. We rank the keywords according to their frequency, and for the five query sets, we omit the top 5%, 6%, 7%, 8%, and 9% frequent keywords, respectively. These experiments aim to evaluate the performance of MAXM-E and MINM-E on queries without highly frequent keywords. We also report the performance of the approximation algorithms for MAX+MAX and MIN+MAX SGK queries.



(a) runtime (b) approximation ratio

Fig. 18. Results of MAX+MAX SGK Queries



(a) runtime (b) approximation ratio

Fig. 19. Results of MIN+MAX SGK queries.

Figure 18(a) shows the runtime of the three approximation algorithms and MAXM-E for the MAX+MAX SGK query. MAXM-E is able to find the answers for all queries within 30 seconds. MAXM-A2 and MAX-AP also run much faster than on the randomly generated query sets. The accuracy of all the approximation algorithms becomes better as well.

Figure 19(a) shows the runtime of MINM-A and MINM-E for the MIN+MAX SGK query. MINM-E is able to find the answers for all queries within two minutes. The accuracy of MINM-A is also better than that on the randomly generated query set.

The two experiments demonstrate that our exact algorithms are able to perform quite well when very frequent keywords are not contained in the queries. Checking dataset Web, we find that most of the frequent keywords are the so-called "stop-words," such as "you" and "very," that do not appear often in queries. Hence our exact algorithms are applicable in most cases.

9.2.5. Scalability. The scalability of algorithms is extremely important in the Big Data era [Cui et al. 2014; Hu et al. 2014]. To evaluate scalability, we generate five datasets containing from 2 to 10 million objects: we generate new locations by copying the locations in GN to nearby locations while maintaining the real distribution of the objects; for each new location, a document is selected randomly from the text descriptions of the objects in GN. Figure 20 shows the runtime of SUM-A and SUM-E for the SUM SGK query, the runtime of MAXM-A1 and MAXM-A2 for the MAX+MAX SGK query, and the runtime of MINM-A for the MIN+MAX SGK query (the number of query keywords is 10). Note that MAXM-A1 and MINM-A are actually the same

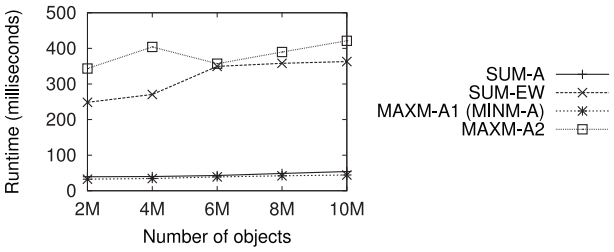


Fig. 20. Scalability of algorithms.

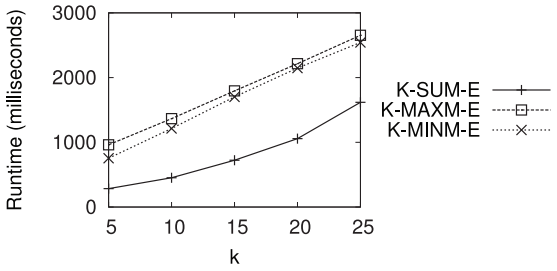


Fig. 21. Runtime.

algorithm. All the algorithms scale well with the size of the dataset. The accuracy changes only slightly and is not affected by the dataset size.

SUM-EN1 and SUM-EN2 run much slower than SUM-E and are omitted. The runtimes of MAXM-E and MINM-E increase exponentially with the dataset size, and are orders of magnitudes slower than the approximation algorithms. Thus, to ensure readability of the figure, we omit them.

9.3. Experimental Results on Top-k SGK Queries

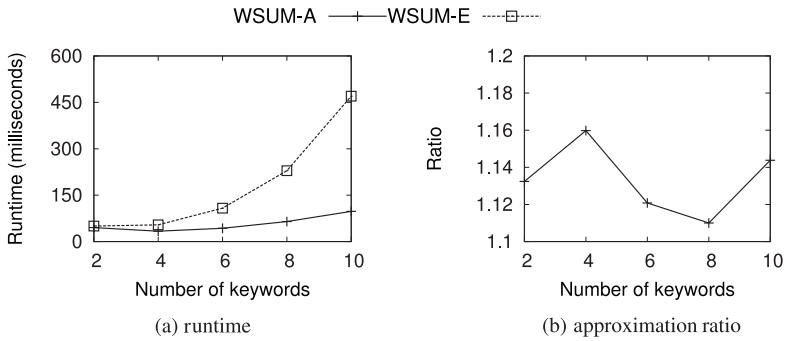
We study the performance of the modified versions of the algorithms for processing top-k SGK queries, that is, the exact algorithm K-SUM-E of the top-k SUM query, the exact algorithm K-MAXM-E of the top-k MAX+MAX query, and the exact algorithm K-MINM-E of the top-k MIN+MAX query. On randomly generated query sets, MAXM-E and MINM-E may fail to give an answer within the 5-minute time limit. Since K-MAXM-E and K-MINM-E are extended from the two algorithms, they also cannot return answers for queries where MAXM-E and MINM-E fail. Thus we conduct this set of experiments on the query set generated as described in Section 9.2.4 by avoiding the top 9% frequent keywords.

The results are shown in Figure 21. We can see that the runtime of K-MAXM-E and K-MINM-E increases linearly with the value of k, and the runtime of K-SUM-E increases a bit faster. This is consistent with the complexity analysis in Section 7.1. The complexity of K-SUM-E is $O(k \log k)$ times that of SUM-E.

9.4. Experimental Results on Weighted SGK Queries

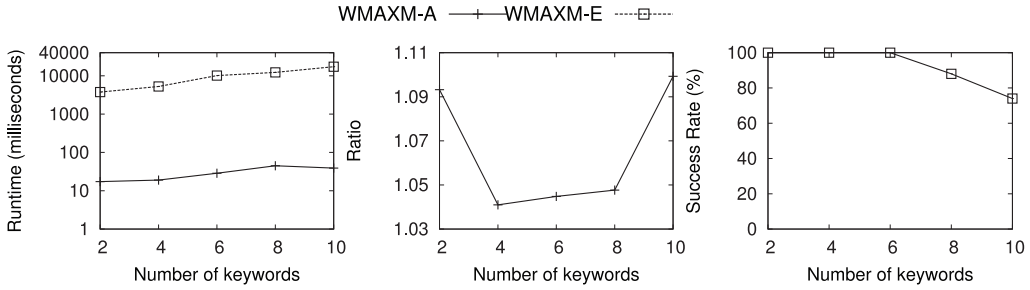
We report on experimental results on Web. We observe similar results on Hotel and GN and thus omit the results for these datasets.

9.4.1. The Weighted SUM *SGK Query.* As shown in Figure 22(a), the approximation algorithm runs much faster than the exact algorithm for the weighted SUM query. Both algorithms take longer time as the number of query keywords increases. Figure 22(b) suggests that WSUM-A can always achieve good accuracy.



(a) runtime (b) approximation ratio

Fig. 22. Results of weighted SUM SGK queries on Web.



(a) runtime (b) approximation ratio (c) success rate

Fig. 23. Results of weighted MAX+MAX SGK queries on Web.

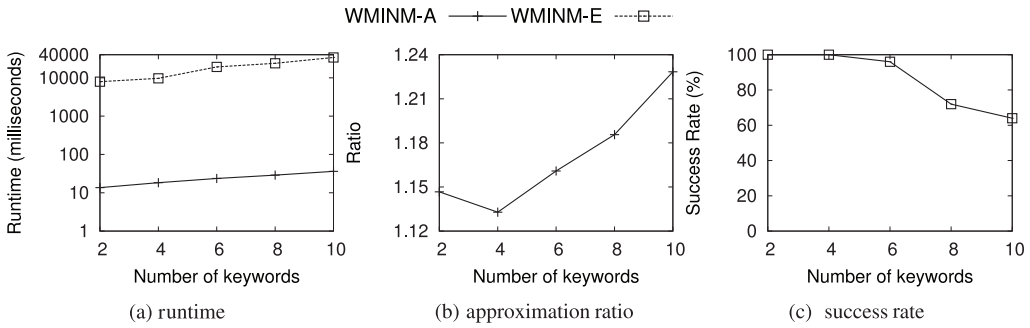
9.4.2. The Weighted MAX+MAX *SGK Query.* Next, Figure 23(a) shows that the approximation algorithm runs much faster than the exact. Since some queries may take too long to compute, we set five minutes as the timeout threshold, as done when evaluating the performance of the exact algorithms for the unweighted query. The success rate is shown in Figure 23(c) and it can be observed that, as the number of query keywords increases, the success rate drops. This occurs because more objects are relevant, which increases the exhaustive search space. Figure 23(b) shows that the actual ratio of WMAXM-A is much better than the worst-case ratio as derived in Theorem 8.2, that is, $1 + 2e$.

9.4.3. The Weighted MIN+MAX *SGK Query.* Figure 24(a) shows the runtime of WMINM-A and WMINM-E. The approximation algorithm runs much faster than the exact algorithm. The success rate is shown in Figure 24(c). The performance of the exact algorithm becomes worse as the number of query keywords increases. Figure 24(b) shows that the actual approximation ratio of WMINM-A is much better than the worst-case ratio as derived in Theorem 8.4, that is, $3e^2$.

10. RELATED WORK

Spatial Keyword Queries. Spatial Web objects are gaining in prevalence, and a number of works on geographical retrieval study the problem of extracting geographic information from Web pages (e.g., Amitay et al. [2004], Ding et al. [2000], and McCurley [2001]), which yields spatial Web objects that can subsequently be queried. This gives prominence to spatial keyword queries [Cao et al. 2012b].

Commercial services such as Google and Yahoo! offer local search functionality. Given a spatial keyword query, they return spatial Web objects, such as stores and



(a) runtime (b) approximation ratio (c) success rate

Fig. 24. Results of weighted MIN+MAX SGK queries on Web.

restaurants, near the query location. The results consist of single objects that each satisfy the query in isolation. In contrast, we aim to find groups of objects such that the objects in a group collectively satisfy a query.

Chen et al. [2013] evaluate the performance of several geo-textual indexes. Several proposals use loose combinations of text indexing (e.g., inverted lists) and spatial indexing (e.g., the R*-tree or the grid index) [Chen et al. 2006; Vaid et al. 2005; Zhou et al. 2005]. They usually employ the two structures in separate stages. In those works either the spatial indexing is first employed and then the text indexing is utilized, or vice versa. Several recently proposed hybrid indexes [Cao et al. 2010; Cong et al. 2009; De Felipe et al. 2008; Hariharan et al. 2007; Khodaei et al. 2010; Rocha-Junior and Nørvåg 2012; Wu et al. 2012a, 2012b; Zhang et al. 2009, 2010] tightly integrate the spatial indexing and the text indexing. In these indexes, each entry p in a tree node stores a *keyword summary field* that concisely summarizes the keywords in the subtree rooted at p. This enables irrelevant entries to be pruned during query processing.

The *IR*^{2}*-tree* [De Felipe et al. 2008] and the *bR*-tree* [Zhang et al. 2009] augment the R-tree with signatures and bitmaps, respectively. Each leaf entry p stores a signature as a fixed-length bitmap that summarizes the set of keywords in p. Each non-leaf entry e stores a signature that is the bit-wise-OR of signatures of the entries in the child node of e. This approach needs to load the signature files of all words into memory when a node is visited, which incurs substantial I/O. Khodaei et al. [2010] propose an index that employs an inverted-file-like structure to store both spatial and text information for objects such that the spatial and textual parts of data can be simultaneously handled. However, it assumes each location as a region and thus cannot be used in our work. Hariharan et al. [2007] propose the KR*-tree (*keyword R*-tree*). Each node of the KR*-tree is virtually augmented with the set of keywords that appear in the subtree rooted at the node. The nodes of the KR*-tree are organized into inverted lists, as are objects. Wu et al. [2012b] propose WIR-tree which aims at partitioning objects into multiple groups such that different groups share as few common keywords as possible.

We use a hybrid geo-textual index, the IR-tree [Cong et al. 2009; Li et al. 2011; Wu et al. 2012a], covered in Section 3, as our index structure.As reported elsewhere [Chen et al. 2013], the IR-tree has good performance and scales well with the dataset size. However, we note that our proposed algorithms are not tied to the IR-tree, but can be used also with the other geo-textual index.

Most existing works on spatial keyword queries retrieve the single object that is close to the query point and relevant to the query keywords. In contrast, we retrieve groups of objects that are close to the query point and collectively meet the keyword requirement.

A few studies consider the retrieval of a group of objects. The mCK query [Guo et al. 2015; Zhang et al. 2009, 2010] takes a set of m keywords as an argument, and

retrieves groups of spatial keyword objects. It returns m objects of minimum diameter that match the m keywords. It is assumed that each object in the result corresponds to a unique query keyword. In contrast, our query takes both a spatial location and a set of keywords as arguments, and its semantics are quite different from those of the mCK query. Bøgh et al. [2013] propose an approach to finding top-k point-of-interest groups given a user requirement in the form of a keyword such as "restaurant". The objects in each returned group are close to the user's current location and to each other. However, all objects in a group are of the same type, making this functionality different from the functionality offered by the SGK query. Cao et al. [2014] propose to retrieve regions of interest for user exploration on road networks. The work aims to retrieve many relevant objects enclosed by a region that satisfies a given size constraint. For example, given query keywords "cafe" and "shopping," it returns a compact region with many objects, each relevant to either "cafe" or "shopping", while the SGK query returns a group that covers all the query keywords with the smallest cost. Long et al. [2013] also propose algorithms to answer our MAX+MAX SGK query. As shown in Section 9.2.2, although their approximation algorithm has better accuracy, its runtime is too high when the number of query keywords increases. Their exact algorithm performs better than a previous proposal [Cao et al. 2011], but worse than the exact algorithm proposed in this article. Note that there is no sensible way to answer spatial group keyword query by splitting the spatial group keyword problem into a spatial and a textual problem, processing them separately, and intersecting their results to get the answer.

This publication builds on our previous work [Cao et al. 2011] in which we propose the spatial group keyword query. In particular, the current article expands that work by the following additional elements: (a) we improve the previous exact algorithm enhanced with the IR-tree in Section 4.2; (b) we design a new approximation algorithm for the MAX+MAX SGK query in Section 5.2, which outperforms the previous approximation algorithm proposed in Cao et al. [2011]; (c) we develop a new exact algorithm for the MAX+MAX SGK query in Section 5.3 which significantly improves the query performance compared with the one in Cao et al. [2011]; (d) we propose a new type of SGK query, that is, the MIN+MAX SGK query, in Section 6 both approximation and exact algorithms are designed to answer this type of query; (e) we propose a new type of query, that is, the top-k SGK (kSGK) query, in Section 7, and we extend the proposed algorithms of answering the SGK queries to process the kSGK queries; (f) we study the weighted spatial group keyword query in Section 8, extend the SUM, MAX+MAX, and MIN+MAX SGK queries to the weighted version, and design approximation and exact algorithms for them; and (g) we present new experimental results for the algorithms in (a)–(f). In addition, we compare our algorithms for the MAX+MAX query with those proposed by Long et al. [2013] in Section 9.2.2, and we analyze the effects of the query keyword frequency on the query processing time in Section 9.2.4.

Route Planning Queries. A route planning query that aims to find a route that meets a user's requirements is related to the SGK query. For example, Li et al. [2005] propose a *trip planning query* (TPQ) in spatial databases and road networks, in which each spatial object has a location and a category. The query aims to find the shortest path between source and target locations that passes through at least one object from each category specified by the user. Sharifzadeh et al. [2008] study a variant of the TPQ [Li et al. 2005], called the *optimal sequenced route* query (OSR). In OSR, a total order on the query categories is imposed and only the starting location is specified. As another example, Cao et al. [2012a] propose the *keyword-aware optimal route* (KOR) search query which finds the route such that it has the smallest objective score and the budget score of the route is smaller than a certain threshold, and meanwhile it

covers all the query keywords. These works aim to find a route instead of a group of objects and thus are different from our SGK query.

Social Group Queries. Lappas et al. [2009] study the problem of finding a team of experts in social networks. The objective is to find a group of persons, each with specific skills, from the social network such that they can collaboratively finish a task and such that their communication cost is minimized. This work does not consider spatial aspects. Yang et al. [2012] propose a social-spatial group query that selects a group of nearby attendees with tight social relations. It is required that the total spatial distance of the attendees to a rally point is minimized while a certain social constraint on the attendees must be satisfied. The cost function is similar to the SUM SGK query. However, they do not consider the textual parts, but instead the social constraint.

11. CONCLUSIONS AND FUTURE WORK

We present the problem of retrieving a group of spatial objects, each associated with a set of keywords, such that the group covers the query's keywords and has the lowest cost measured by their distance to the query point as well as the distances between the objects in the group. We study three particular instances of the problem, all NP-hard. We develop approximation algorithms with provable approximation bounds and exact algorithms to solve the problems. We also study the top-k spatial group keyword query and the weighted version of the query. Results of experimental evaluations offer insight into the efficiency and accuracy of the approximation algorithms, and the efficiency of the exact algorithms.

This work opens a number of promising directions for future work. First, it is of interest to develop algorithms for the SUM+MAX SGK query. Second, it is of interest to consider the problem of a partial coverage of query keywords. That is, if the cost of covering all query keywords is too high, we can consider to partially cover the query keywords to reduce the cost.

REFERENCES

Einat Amitay, Nadav Harel, Ron Sivan, and Aya Soffer. 2004. Web-a-where: Geotagging web content. In *Proceedings of the 27th Annual ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR'04).* 273–280.

Esther M. Arkin and Refael Hassin. 2000. Minimum-diameter covering problems. *Netw.* 36, 3, 147–155.

Franz Aurenhammer and Herbert Edelsbrunner. 1984. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Patt. Recogn.* 17, 2, 251–257.

Kenneth Bøgh, Anders Skovsgaard, and Christian S. Jensen. 2013. GroupFinder: A new approach to op-k point-of-interest group retrieval. *Proc. VLDB Endow.* 6, 12, 1226–1229.

Xin Cao, Lisi Chen, Gao Cong, Christian S. Jensen, Qiang Qu, Anders Skovsgaard, Dingming Wu, and Man Lung Yiu. 2012b. Spatial keyword querying. In *Proceedings of the 31st International Conference on Conceptual Modelling (ER'12).* 16–29.

Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. 2012a. Keyword-aware optimal route search. *Proc. VLDB Endow.* 5, 11, 1136–1147.

Xin Cao, Gao Cong, and Christian S. Jensen. 2010. Retrieving top-k prestige-based relevant spatial web objects. *Proc. VLDB Endow.* 3, 1, 373–384.

Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11).* 373–384.

Xin Cao, Gao Cong, Christian S. Jensen, and Man Lung Yiu. 2014. Retrieving regions of interest for user exploration. *Proc. VLDB Endow.* 7, 9, 733–744.

Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial keyword query processing: An experimental evaluation. *Proc. VLDB Endow.* 6, 3, 217–228.

Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*. 277–288.

Vaclav Chvatal. 1979. A greedy heuristic for the set-covering problem. *Math. Oper. Res.* 4, 233–235.

Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endow.* 2, 1, 337–348.

Bin Cui, Hong Mei, and Beng Chin Ooi. 2014. Big data: The driver for innovation in databases. *Nat. Sci. Rev.* 1, 1, 27–30.

Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword search on spatial databases. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE'08)*. 656–665.

Junyan Ding, Luis Gravano, and Narayanan Shivakumar. 2000. Computing geographical scopes of web-bresources. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*. 545–556.

Tao Guo, Xin Cao, and Gao Cong. 2015. Efficient algorithms for answering the m-closest keywords query. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*. To appear.

Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'84)*. 47–57.

Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. 2007. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM'07)*. 16.

Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. 2014. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access* 2, 652–687.

Ali Khodaei, Cyrus Shahabi, and Chen Li. 2010. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA'10)*. 450–466.

Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. 467–476.

Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On trip planning queries in spatial databases. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases (SSTD'05)*. 273–290.

Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Engin.* 23, 4, 585–599.

Cheng Long, Raymond Chi-Wing Wong, Ke Wang, and Ada Wai-Chee Fu. 2013. Collective spatial keyword queries: A distance owner-driven approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. 689–700.

Kevin S. McCurley. 2001. Geospatial mapping and navigation of the web. In *Proceedings of the 10th International Conference on World Wide Web (WWW'01)*. 221–229.

Joao B. Rocha-Junior and Kjetil Nørvag. 2012. Top-k spatial keyword queries on road networks. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*. 168–179.

Mehdi Sharifzadeh, Mohammad R. Kolahdouzan, and Cyrus Shahabi. 2008. The optimal sequenced route query. *VLDB J.* 17, 4, 765–787.

Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. 2005. Spatio-textual indexing for geographical search on the Web. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases (SSTD'05)*. 218–235.

Dingming Wu, Gao Cong, and Christian S. Jensen. 2012a. A framework for efficient spatial web object retrieval. *VLDB J.* 21, 6, 797–822.

Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. 2012b. Joint top-k spatial keyword query processing. *IEEE Trans. Knowl. Data Engin.* 24, 10, 1889–1903.

Dingming Wu, Man Lung Yiu, Christian S. Jensen, and Gao Cong. 2011. Efficient continuously moving top-k spatial keyword query processing. In *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE'11)*. 541–552.

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 *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'12)*. 949–957.

Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.* 22, 2, 179–214.

Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. 2009. Keyword search in spatial databases: Towards searching by document. In *Proceedings of the 25th International Conference on Data Engineering (ICDE'09)*. 688–699.

Dongxiang Zhang, Beng Chin Ooi, and Anthony K. H. Tung. 2010. Locating mapped resources in Web 2.0. In *Proceedings of the 26th International Conference on Data Engineering (ICDE'10)*. 521–532.

Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. 2005. Hybrid index structures for location-based web search. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM'05)*. 155–162.

Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *ACM Comput. Surv.* 38, 2, 6.

Received October 2013; revised March 2015; accepted March 2015