2019 2nd International Conference on Mechanical Engineering, Industrial Materials and Industrial Electronics (MEIMIE 2019)

TP-SKQ: An Efficient Index for Location Aware Rank Query

Bohong Liu, Xin Yi[a]

School of Chongqing University of Posts and Telecommunications University, Chongqing 400000, China

[a]826348054@qq.com

Keywords: Tp-skq, wireless communication technology, ranking query algorithm

Abstract: With the rapid development of wireless communication technology and the wide application of Intelligent Mobile Terminal, a large amount of geographic text data is generated every moment. How to deal with location-aware ranking queries efficiently has become an urgent problem. In this paper, a hybrid index structure *tp-skq* is proposed which supports user preference constraints. And based on it, an efficient location-aware ranking query algorithm is coming up.

1. Introduction

With the rapid development of wireless communication technology and the wide application of Intelligent Mobile Terminal, the integration of geographic location and text data is becoming more prevalent. Therefore, an increasing number of Web objects have both location attributes and text description. For example, the content published by users is usually closely related to their location in micro-blog and WeChat. In the application of sign, users will automatically record their location coordinates while checking in. At the same time, users often use words to describe their moods and feelings. With the rapid development of location-based services (*lbs*), more and more location-aware data sets are created and used. How to deal with location-aware rank query (*lrq*) efficiently is urgently tackled [1].

Traditional *lrq* included nearest neighbor query (*knn*) [2] and spatial keyword query (*skq*) [3]. They are widely used in many fields. However, with the introduction of many new data types, such as numbers, time and so on, *lrq* become more and more complex. Users can find the nearest and most relevant objects through spatial keyword query. However, with the development of personalized services, users are more likely to query objects that satisfy their preferences. For example, when inquiring about restaurants, users may have preferences for taste, service, hygiene, etc. Sometimes, this preference can be expressed by keywords, and sometimes by numerical values. Take an example, assuming that restaurants set a score for service level and hygiene level, this preference can be expressed as "service > 8.0" or "hygiene > 7.5". How to deal with these preferences making the query results be better meet the users' needs is a very meaningful research issue. It was the first to study this issue by document [4]. It indexed users' preference attributes by constructing synopses tree and combined with IR-tree [5]. When querying, whether they intersect with preference query intervals good or not is a judgement to the pruning effect. On this basis, a spatial keyword *top-k* query processing framework *linq* with user preference constraints is proposed. However, in real life, preference attributes may fluctuate greatly in the short term due to some changes, such as the replacement of chefs and waiters in catering and accommodation industries, and the renovation of the environment. Therefore, in recent years, the attribute query will be more in line with and better meet the needs of users. How to process these time-marked preference attribute datasets and establish an efficient index structure to enable users to query location-aware objects with Preference Constraints similar to recent taste and service > 8 is the key to solve the problem.

According to the method proposed in document [4], it also can be used to process preference attribute data sets with time attributes after appropriate modification. However, there are some shortcomings: time is treated as a common numerical attribute, and it is added to preference attribute data sets to construct a synopses tree to deal with time attributes. As the dimension of the synopses tree is increased, the estimation error rate increases, and more leaf nodes need to be accessed, which affects the query processing performance. In order to solve those problems, this paper proposes to take the time attribute as the time axis and construct a synopses tree under each time stamp, that is, to construct an index structure based on *temporal and preference query* (*tpq*). By associating *tpq* with IR-tree, constructing hybrid index structure *tp-skq* and corresponding optimization query algorithm, *lrq* with user preference constraints can be effectively processed.

2. Related work

2.1 Section Headings

In recent years, with the popularity of *lbs*, the use of given query locations and query keywords to search related geographic text objects has attracted great attention [6, 7, 8]. In order to solve the *lrq* problem, many indexes have proposed the method of combining spatial and text information in the same data structure, such as IR-tree [5] and its variants. IR-tree associates an inverted file with each node in R-tree. The inverted file records where each keyword appears in the sub-nodes. WIR-tree [9] is a variant of IR-tree. It differs from IR-tree in that it aggregates objects in different ways. WIR-tree groups objects according to the keywords contained, so that the keywords shared by each group are as few as possible. BR-tree [10] indexed the text information by associating bitmaps of each node of the R tree. When querying, it pruned the nodes of the R tree according to whether the bitmap contained all the query keywords. Finally, it sorted the objects according to the distance proximity. However, these studies only focus on the query processing of spatial text data. When dealing with user preference constrained *lrq*, only all the information of the candidate result set can be obtained, and the final result set can be determined by judging whether it satisfies the preference query. The query efficiency is low. This paper focuses on *lrq* processing with user preference constraints.

Many researchers have contributed to the problem of user preferences for querying a geo-text object. Document [11] proposes a multi-IRS index structure, which is a multi-tree index that can improve the index structure optimization based on a given query at runtime. It solves the problem that the hybrid index of IR-tree and synopses tree in document [5] can’t realize the optimization performance because the index creation process is consistent with the traditional method. Document [12] proposes an optimization technique for text attributes that can be included in Multi-IRS, which effectively reduces the cost of I/O access. Because the current research on preferences is still at its primary stage, the existing research results can’t meet the more complex and targeted needs of the public. In view of the above shortcomings, this paper proposes a hybrid index structure.

3. Design and Analysis of TP-SKQ Hybrid Index Structure

3.1 Description of the problem

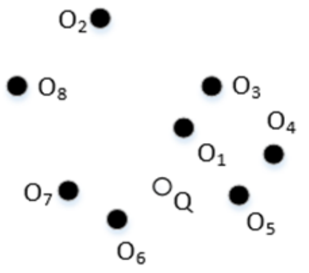
When we considering the time attribute, it is assumed that the position-aware object o is represented as a triad, which concludes λ,*w* and *l*. Then *o.*λ represents the position of the object, *o.w* represents a set of keywords, and *o.l* represents a set of numerical preference attributes. *o.l* contains the numerical preference attribute *o.p* and the time attribute *o.t* of *o*. Each value preference attribute data in *l* is distinguished by time, and $l = \{o_1t_1p_1, o_1t_1p_2, \ldots o_it_jp_k \ldots\}$.

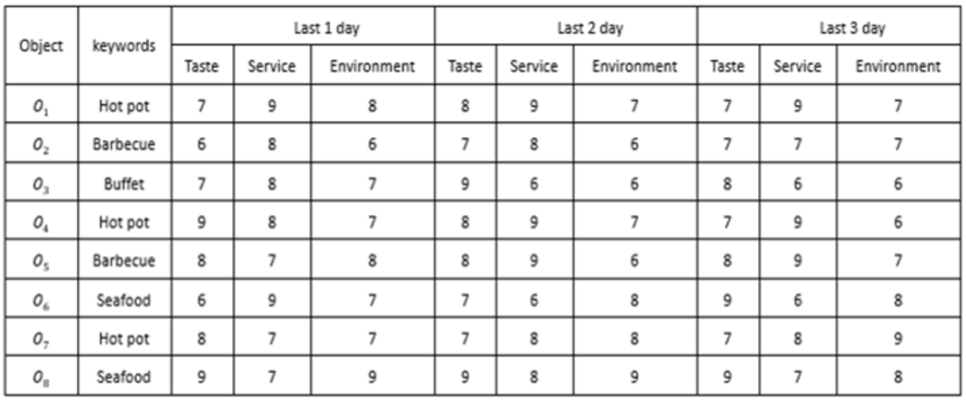
The location-aware rank query based on the preference constraints (*lrpq*). Formally, a query *q* is a quintuple, $q= (\lambda, w, c, f, k)$. Then $q.\lambda$ represents the positional description of *q*, *q.w* the set of query keywords and *q.c* a set of numerical preference constraints. *q.f* is considered as a sorting function. *q.k* is the number of results returned.

The required conditions of the query. A set of location-aware objects d and a query *q* which is based on preference constraints are given, to inquire *q* of all or *q.k* of its location-aware objects from the set of objects d to ensure that each object meets the constraints in the query of *q.c*. Then the output is sorted according to the formula *q.f*. The sorting formula is defined as follow:

$$\mathrm{f}(o,q)=\begin{cases}0 & \neg q(o)\\ g(prox(o,q),rel(o,q), & \\ \quad s_1(o.p_1),s_2(o.p_1),\dots) & otherwise,\end{cases} \tag{1}$$

In this formula, $\neg q$ *(o)* indicates that the location-aware object o does not satisfy the query constraint *q.c. prox(o,q)* is used to calculate the distance between the object location $o.\lambda$ and the query location $q.\lambda$. *rel(o,q)* is used to calculate the relativity between the keyword set do and the query keyword *q.w.* s_1 *(o.*p_1 *)*,s_2 *(o.*p_1 *)*,…represents the preference attribute. *o.*p_1 *,o.*p_2,… represents the scoring function. g is a monotonic function which adopts the linear combination function.





Object	keywords	Last 1 day			Last 2 day			Last 3 day		
		Taste	Service	Environment	Taste	Service	Environment	Taste	Service	Environment
o_1	Hot pot	7	9	8	8	9	7	7	9	7
o_2	Barbecue	6	8	6	7	8	6	7	7	7
o_3	Buffet	7	8	7	9	6	6	8	6	6
o_4	Hot pot	9	8	7	8	9	7	7	9	6
o_5	Barbecue	8	7	8	8	9	6	8	9	7
o_6	Seafood	6	9	7	7	6	8	9	6	8
o_7	Hot pot	8	7	7	7	8	8	7	8	9
o_8	Seafood	9	7	9	9	8	9	9	7	8

Figure 1. A set of location-aware objects.

3.2 TP-SKQ hybrid index structure

First, based on the synopses tree, the *tpq* index structure is proposed to process the numeric attributes of the user preferences with time attributes. Then, combined with the spatial keyword index, a hybrid index structure *tp-skq* supporting spatial position, text and numerical attributes of user preferences with time attributes is proposed.

3.2.1 Design of the TPQ index structure

The *tpq* index structure is applied to index the numeric attributes of user preferences with time attributes. The construction of the temporal summary tree is roughly processed in two steps. At first, a timeline is to be built in chronological order. Then a tense-based profile tree is to be built. Namely, the tpq index structure. For example, to construct a *tpq* index structure for the data set in Fig.1. The data set gives a list of preferred value attributes for the last three days. Firstly, a timeline with timestamps (t) of 1, 2, 3 in chronological order is to be constructed. Then a profile tree is to be created under each timestamp. That is, when the timestamps t=1, 2, 3, three summary trees s_1, s_2, and s_3 are respectively established. The construction of the summary tree is roughly divided into three steps.

The factorization histogram. Firstly, as is shown in Fig.2 (a), a model of the moral map is to be built. In this model, each of these nodes represents an attribute, and each side represents a relationship between a pair of attributes. Then, a connection tree is to be created based on the ethical graph. Put the attributes together with the attributes which the former depends on. The connection tree is shown in Fig.2 (b). According to the connection tree, distributions of the taste rating and the service rating and the taste rating and the environmental rating is reserved. Other distributions can be derived from these two distributions.

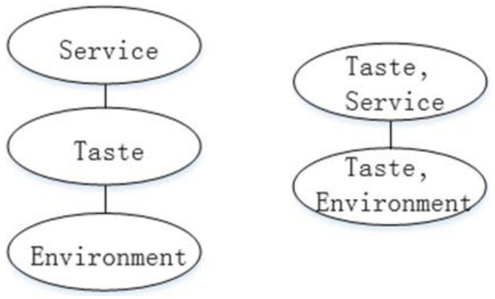
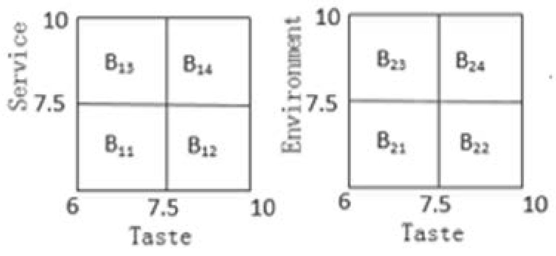
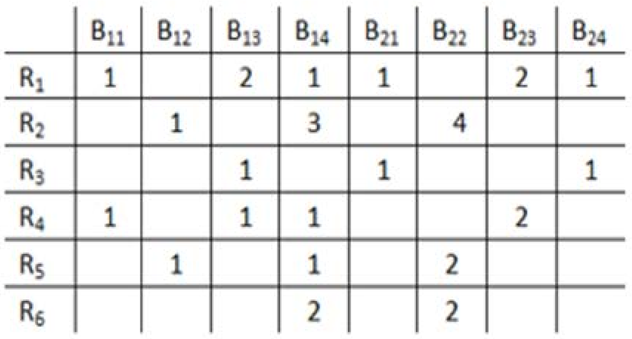


Figure 2. An example of junction tree. (a) The moral graph, (b) the junction tree.

The construction of a global histogram. Construct a set of global histograms under the time axis *th*={t_1h_1,t_1h_2,…t_nh_k } for the entire preference attribute data set, in which *tihj*={*b*j1,*b*j2, …} is a global histogram under the time stamp t_i and b_{jk} is a histogram of a bucket. The collection of global histograms is used by all entries in the profile tree. Let b be the set of buckets in all histograms, *b*= $\cup t_ih_j$, and |b| is the number of buckets in *b*. For any entry e, ensure there is an element |b| in an array e.b , where each element in the array is a statistic about *e.d* in a bucket. In this way, the array records the summary statistics of e.d . For example, a histogram of the two-dimensional data set (flavor score, service score) and (flavor score, environmental score) is constructed at timestamp t=2, as is shown in Fig.3 (a) and (b). It is assumed that eight buckets are available. t_2h_1 and t_2h_2 are respectively the global histograms of (flavor score, service score) and (taste score, environmental score). Fig.3 (c) shows the local information held by each entry, where R_1, ..., R_6 are the entries in the R-tree.





	B_{11}	B_{12}	B_{13}	B_{14}	B_{21}	B_{22}	B_{23}	B_{24}
R_1	1		2	1	1		2	1
R_2		1		3		4		
R_3			1		1			1
R_4	1		1	1			2	
R_5		1		1		2		
R_6				2		2		

(a) (b) (c)

Figure 3. The histograms constructed for the example dataset. (a) t_2h_1, (b) t_2h_2, (c) local information.

Use bit to represent the local information. In the example above, for each entry, counting information about each bucket is kept. In fact, it is enough to count some simple statistics since we only care about the satisfiability of the query of predicates in this work. Therefore, we use a compact representation of bit-based local information in each bucket and each partition is split into m partitions by successive two-dimensional partitioning in a round-robin fashion, and then an m-bit string is stored. If the corresponding partition is not empty, each bit is 1, if it, it is 0. Since the object instance in Fig.1 contains fewer attribute data sets, m=1 is used in this example to indicate whether the bucket is empty. As is shown in Fig.4, in order to describe the data distribution of the entry R_4 with respect to b_{11}, a 1-bit character string (m=1) is used, and since the corresponding partition is not empty, it is denoted as 1. The synopses in R4 under the timestamp t=2 is an array of bit strings corresponding to the buckets in the global histogram *th*. Therefore, the synopses of an entry is an array of strings corresponding to the buckets in the global histogram.

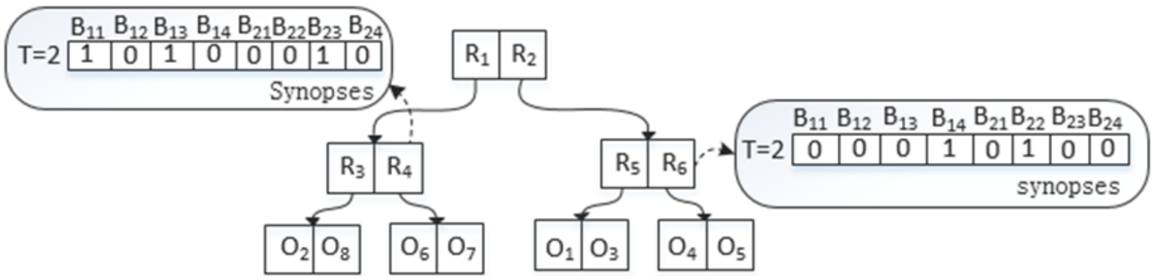


Figure 4. A synopses tree of the timestamp t = 2

As is shown in Fig.5, the structures of s1, s2 and s3 are all related to the same R-tree. The nodes in the R-tree are respectively associated with the profiles in the three synopses trees. Each of the profiles records the distribution of the attribute dataset corresponding to the node under the timestamp. The structure of the *tpq* index is related to the R-tree associated. Suppose there are *no* timestamps, then each node stores n synopses tree under different timestamps and represents the dataset distribution of the R-tree nodes associated. For each entry in a non-leaf node in the R-tree, these n outlines are to be maintained to save information about buckets in n global histograms. An R-tree entry is associated with n profiles. Each of the profiles is an array of bit strings corresponding to the buckets in the global histogram at the corresponding timestamp.

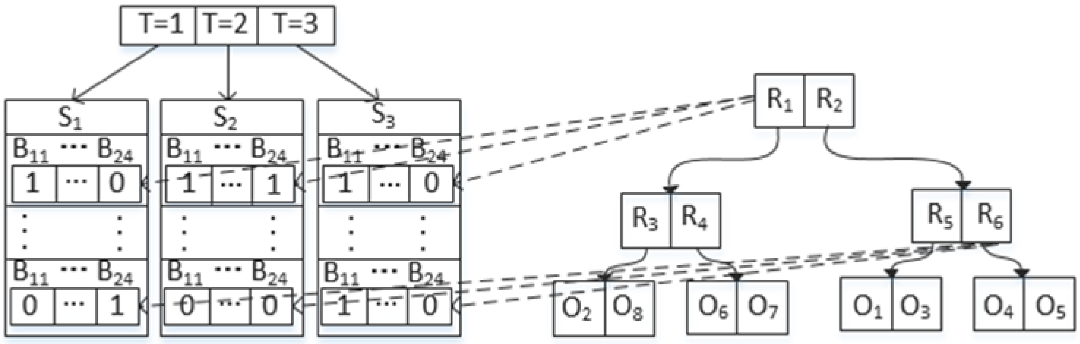


Figure 5**.** The *tpq* index structure diagram

3.2.2 The combination of the tpq index and the spatial keyword index

The *tpq* index structure summarizes the distribution of numerical preference properties based on tenses. It can solve some of the location-aware query and processing problems based on preference constraints. For example, a synopses tree can be used to find objects that meet the demands of service≥8∩taste≥8 in the past two days. In order to solve the problem of location-aware query processing based on preference constraints, the *tpq* index structure must be combined with other indexes both locational and textually.

The *tpq* can be easily combined with the IR-tree [5]. The two is to be stored separately. The IR-tree can be built formally. Then a global histogram based on the tense is to be built. Fig.6 shows the *tp-skq* hybrid index structure constructed by the object instance shown in Fig.1. The R-tree lies in the center of the graph to index the position of the object. Each node in the R-tree, in addition to the pointer field that points to the child node and all the circumscribed rectangles that are recorded in the child node, is associated with two additional components: the synopses of the document in the IR-tree and the synopses of the different timestamps in the *tpq*. The inverted files and the like, the synopses and the R-tree are stored separately.

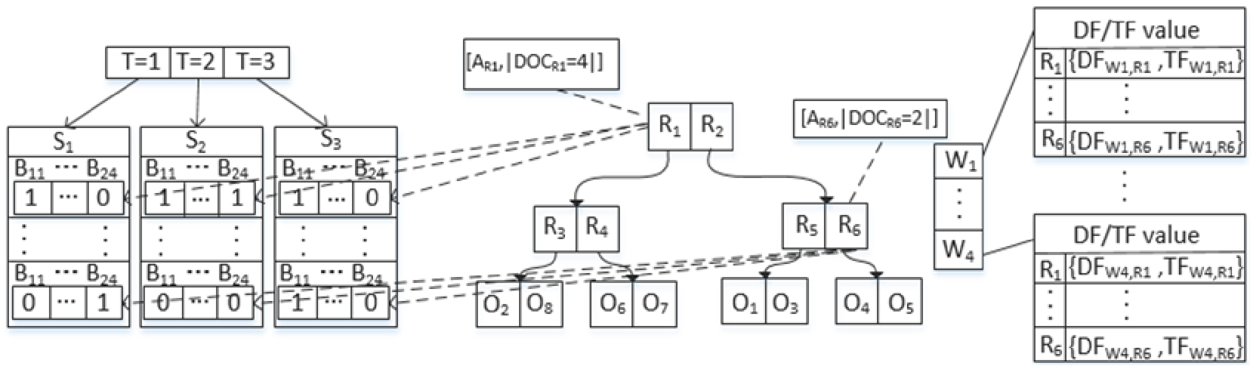


Figure 6. *Tp-skq* hybrid index structure diagram

3.3 Updating and Maintenance of the Index Strucure

3.3.1 Insertion Algorithm

Algorithm 1 describes the insertion algorithm. First, find the appropriate inserted leaf node *n* for the object *o*, and then add o to the leaf node n and update the summary and inverted files of the node n (line1-4). If there is no extra space in the node to store *o.λ*, split node n and calculate the summary and inverted files of the two nodes. In line 7, the synopses of nodes *n* and *nn* is to be calculated. Each node contains multiple profiles separated by timestamps. *n.syn* and *nn.syn* are calculated based on their entries. Then determine if n is the root node. The summary of *n* is to be updated, namely, some bits under each timestamp are changed from 0 to 1. The update is trivial because the new data only affects a few partitions.

Table 1. Insertion algorithm.

```
Algorithm 1.Insert (O)
(1) n→ChooseLeaf(o.λ)
(2) Add o.λ to n
(3) Update n.syn
(4) Update n.InvFile
(5) If n don't have extra space Then
(6)   {n,n}←SplitNode(n)
(7)   Compute n.syn and nn.syn
(8)   Compute n. InvFile and nn. InvFile
(9)   If n is root then
(10)     Initialize a new node n0,let n and nn be children of n0,and set n0 be the new root
(11)     Else
(12)     Ascend from n to the root, adjusting the covering mbrs, updating synopses and propagating node splits as necessary
(13) end if
(14) else if n is not root then
(15) Update n.syn
(16) Update n. InvFile
(17) end if
```

3.3.2 Deletion Algorithm

Algorithm 2 describes the deletion of the algorithm. Firstly, assign the root node of the tree to node *n* (line 1). Then find out the entry *e* containing *o.l* in loop *n*. Determine if *e* is a leaf node, if not, assign *e* to *n*. Repeat from line 2 to find the leaf node containing *o. λ*. *o. λ* is then removed from the found leaf nodes. The leaf node's summary and inverted files are updated. Finally, adjust the tree (Concentrate Tree method), use the linked list *q-list* to store the entries contained in the deleted node (line 1 of the Concentrate Tree method). Determine if n is the root node (line 2 of the Concentrate

Table 2. Deletion algorithm.

```
Algorithm 2. Delete(O)
(1) n←root of R-tree
(2) Foreach e in n[index] Do
(3)   If e.mbr intersects with o. λThen
(4)    If e is not leaf node;
(5)      n←e Then goto(2) ;
(6)    Else
(7)      break;
(8)    End If
(9)   End if
(10) End
(11) When E is not leaf node, terminate the whole algorithm;
(12) Delete o. λfrom e;
(13) Update e.syn;
(14) Update e. InvFile;
(15) ConcentrateTree (e);
(16) Adjust root of tp-skq;
ConcentrateTree (n)
(1) q-list←NewQueueList;
(2) If n.Level is tree.Height Then
(3)         Goto(17);
(4) Else
(5)    p is parent of n;
(6)    en←p point to n;
(7)    If n.entryCount m Then
(8)        Delete en from p;
(9)       q-list←all the entrys in n;
(10)      Else
(11)        Adjust N.MBR to surround n.entrys;
(12)        End If
(13)        Update p.syn;
(14)        Update p.syn;
(15)        Set n←p; goto(2);
(16) End If
(17) ReInsert all remain entries of q-list
```

Tree method). If it is, re-insert the nodes in the *q-list* into the tree, where the entries originally contained in the leaf nodes are reinserted with the insertion algorithm. For those entries originally included in the non-leaf nodes, they must be inserted into the nodes of the layer where the non-leaf nodes are before the deletion to ensure that the entries pointed to by the non-leaf nodes are still in the same layer in the R-tree (line 5-6 of the Concentrate Tree method). If *n* is not the root node of the tree, then the parent node P of *n* is to be found, and let *en* be a pointer to *n* stored in the *p* node (line 17 of the Concentrate Tree method). When the number of entries contained in n is less than the minimum threshold *m*, *en* is to be deleted from p, and all entries in n are to be inserted in the *q-list*. If n is not deleted, then adjust *en.i* so that it can just cover the *mbr* of all entries in n (line 7-12 of the Concentrate Tree method). Update the synopses tree and inverted files of the parent node p (lines 13-14 of the Concentrate Tree method). Let *n* be equal to *p* and then repeat the operation from line 2 of the Concentrate Tree method (line 15 of the Concentrate Tree method). Compress the tree. When there is only one child node in the root node, set this child node as a new root node and delete the original root node (line 16 of the Concentrate Tree method).

3.3.3 Modification Algorithm

Algorithm 3 describes the modification of the algorithm. The modification operation on the *tp-skq* hybrid index structure is related to the change of the location information $o.\lambda$, the text information *o.w* and the numerical attribute information *o.p* of the location-aware object *o*. When it comes to changing $o.\lambda$, it is actually a combination of insert operation and delete operation. When $o.\lambda$ is modified, it will definitely change the minimum enclosing rectangle in its node, that is, deleting the original record information of the object *o* from the tp-skq and inserting the modified record. When the $o.\lambda$ is not to be modified, assign the root node of the tree to node n (line 5). Then recycling the entries in loop n to find out the entry *e* containing *o* (line 6-7). If o.w is modified, then update the inverted file information in e (line 8-10). If *o.p* is modified, update the summary in e (line 11-13). Finally, determine whether e is a leaf node. If it is not, assign e to n and repeat this process from line 2 to find out the leaf node containing $o.\lambda$ (line 14-20). Then find o and modify *o.w* and *o.p* (line 21-28).

Table 3. Modification algorithm.

```
Algorithm 1. modify (O)
(1)n←root of R-tree;
(2)Foreach e in n[index] Do
(3)If e.mbr intersects with o. λ Then
(4)     If modify o.w Then
(5)       Update e.InvFile;
(6)     End
(7)     If modify o.p Then
(8)       Update e.syn;
(9)     End
(10)    If e is not a leaf node;
(11)      n←e Then goto(6) ;
(12)    Else
(13)      break;
(14)    End If
(15) End if
(16) End Foreach
(17) Find o in e [index];
(18) If modify o.w Then
(19)     Update o.w;
(20) End
(21) If modify o.p Then
(22)    Update o.p;
(23) End
```

3.4 Query Algorithm

Firstly, initialize the *top-k* to maintain the current result of the *top-k*. Put the root node root into the priority queue *pq* (line 1-2). According to the timestamp in the query predicate, select the global histogram h and the connection tree under this timestamp (line 3-4). Then determine if the priority queue *pq* is empty. When the *pq* is not empty, the first element in the *pq* is to be taken out and the matching score in it is to be assigned to *do*, and the object or node is assigned to n (line 5-6). Finally, each node *n* is to be determined. If *n* is the object, insert it into the *top-k*, and if the result of the *top-k* is *q.k*, then return to the *top-k* (lines 7-11). If *n* is an internal node instead of a leaf node, then for each entry *e* inn, based on the timestamp in the query predicates, select the summary under the timestamp in *e*. Then estimate *maxf (e,q)* on the basis of this summary. If the *maxf (e, q)* is greater than 0, it means that there may be objects in the entries contained in e that satisfy the query predicate. Thus, *e* and *maxf (e, q)* are added to the queue *pq*. *pqs* are sorted by matching scores (line 12-20).

In the algorithm, the priority queue pq is used to track nodes and objects and is arranged in descending order of scores. The result set is stored in top-k. F (e, q) is calculated according to Equation 1 and maxf (e, q) Equation 2.

$$\max f\,(n,q) = \begin{cases} 0 & , q(n) = \emptyset \\ max(g(prox(e,q), rel(e,q), s_1(e.p_1), s_2(e.p_2), \dots), & otherwise \end{cases} \quad (2)$$

Q (n) represents a set of objects satisfying the query predicate surrounded by the node n. *prox* (*e*, *q*) is the spatial proximity of *mbr* and $q.\lambda$ corresponding to *e*. prox (*e*, *q*) is represented by the reciprocal of the shortest distance between $q.\ \lambda$ and *mbr* of entry *e*. *rel (e, q)* represents the textual similarity between the document set do and *q.w*. And *e.pi* represents the score calculated by the numerical attribute *p* in *e.d*. And *s*1 (.),s_2(.),… represents the scoring function of attributes $e.p_1$, $e.p_2$,....Among them, $e.p_i$ represents the p_i attribute of all objects in the item *e*. The maximum value is to be taken in this calculation.

Table 4. Query algorithm

```
Algorithm 4.
(1) top-k is null;
(2) pq.push (Root,0);
(3) select the junction tree j and global histograms h by q.c;
(4) rds(root)←Precomputation (q, root, j, h);
(5) While pq is not null Do
(6)   n and d←pq.pop();
(7)   If n is an object Then
(8)     topk.insert(n);
(9)     If top-k.count equals q.k;
(10)      break;
(11)    End If
(12)  Else If n is not a leafNode Then
(13)    Foreach e in n[index] Do
(14)      select the e.synopse by q.c;
(15)      maxf(e,q)←Estimate(e);
(16)      d←maxf(e,q);
(17)      If d is greater than 0 Then
(18)        pq.push(e,d);
(19)      End If
```

4. Experimental Results and Analysis

4.1 Experimental Environment

Experiments were carried out under 64-bit Microsoft Windows 7 operating system. All experiments were carried out using C # development language and Visual Studio 2010 development tools. Hardware environment: Inter (R) Core (TM) i5-2450M CPU@2.50GHz, 4GRAM. The experimental data set of this experiment uses real data and synthetic data. The real data is the non-private data (business name, address, cuisine, taste, environment, etc.) of the gourmet merchants published by the public comment website. After processing, it contains 50,000 objects with geographic location and text attributes (name, keyword, etc.) and three numerical attributes (taste, service, and environment). Composite data is attribute data set, which is used to generate attribute data (taste, service, environment, time) for d days. The values of each attribute are generated randomly and independently, in order to better verify the good query performance based on tp-skq. Because the main difference between tp-skq and improved IR-synopses tree is the processing of attribute data, all experiments are carried out under the condition that the number of objects is 50,000.

4.2 Experimental Result.

In this experiment, under different parameters, 100,000 attribute data sets, 300,000 attribute data sets and 600,000 attribute data sets under 50,000 objects are used to test the performance of *tp-skq* index. Queries involve location, keywords, preference attributes, and time. The preferences pruning query based on *tp-skq* is compared with the query based on improved IR-synopses tree.

(1) Index performance comparison

TP-SKQ and improved IR-synopses trees are constructed on test datasets with different attribute data quantities. As shown in Fig.7, the build times of the two are quite different. In different size attribute datasets, the time of building an IR-synopses tree based on *tp-skq* is significantly shorter than that of building an IR-synopses tree. Moreover, with the increase of the amount of attribute data, the performance advantage of *tp-skq* is more obvious.

The difference of the construction time of the two hybrid index structures is mainly reflected in the time of constructing global histogram. In fact, according to the particularity of temporal attributes, the process of constructing global histogram in *tp-skq* is to reduce the dimension of global histogram constructed in IR-synopses tree, and to group attribute datasets according to time, so as to avoid using all attribute datasets to construct histogram at one time, so the construction time gap between the two index structures is becoming more and more obvious.

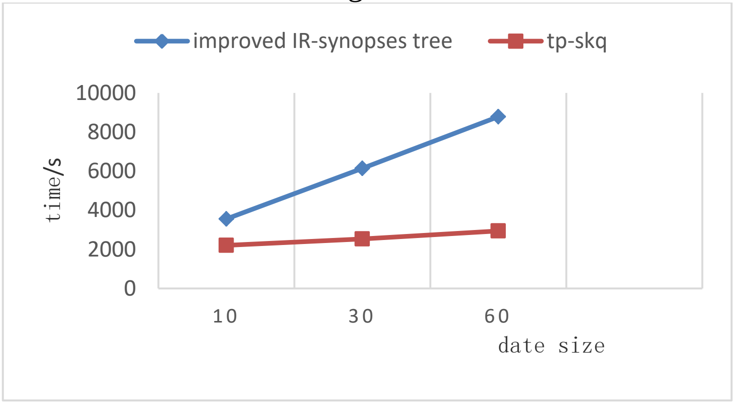


Figure 7. Index structure performance contrast

(2) Comparison of query time under different data quantities

When other parameters are default values, the query time of two different index structures varies with the size of attribute data as shown in Fig.8. The query time under two different index structures increases with the increase of the amount of attribute data. Under different attribute dataset sizes, summary pruning queries based on *tp-skq* are more efficient than those based on IR-synopses tree, and the gap increases with the increase of the amount of attribute data. This is due to the fact that the temporal profile tree is used to estimate the pruning time and the estimation error rate in the query process of the summary pruning query based on *tp-skq*, so the query time is reduced. Moreover, with the increase of attribute data, the estimation error rate of query based on IR-synopses tree will increase gradually in the query process, so the query time will increase relatively fast. This demonstrates the superiority of the performance of summary pruning query based on *tp-skq*.

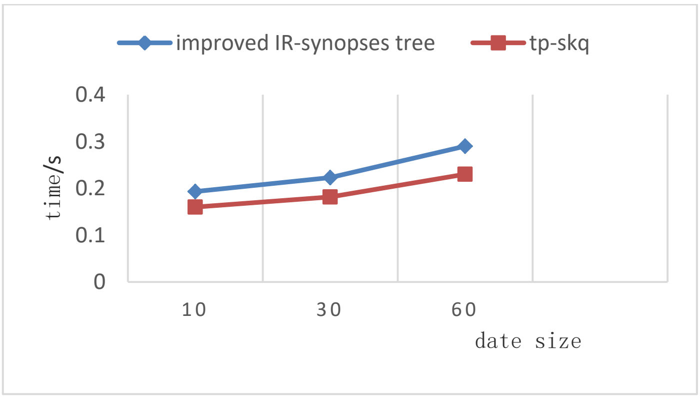


Figure 8. The query time contrast

(3) Comparison of query time under different *k* values

When other parameters are default values, the query time of two different index structures under different *k* values is shown in Fig.9. With the increasing number of requests k, the query time spent on the *tp-skq* pruning query and the IR-synopses tree query is increasing. The reason is that as the number of search result sets increases, more nodes or objects need to be estimated, so the time increases. Because the query time of the proposed base on the *tp-skq* index preference pruning query is always shorter than that of the IR-synopses based query under any *k* value, it can be concluded that the *tp-skq* index preference pruning query has better query performance.

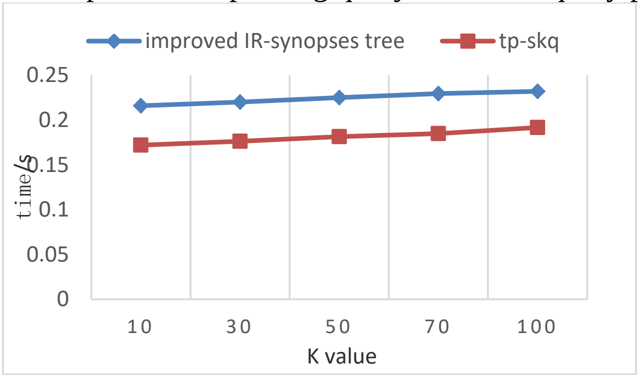


Figure 9**.** K value change of the contrast

(4) Coverage of different predicates

When the amount of attribute data is 300,000, the query time of two different index structures under different predicate coverage is shown in Fig.10. With the increasing coverage of predicates, the query time spent on both *tp-skq* pruning queries and IR-synopses tree queries is increasing, and the gap between them is getting smaller and smaller. The reason is that with the increase of predicate coverage, the satisfiability of node profiles increases and the number of searchable nodes increases, so both of them increase in time. It is also due to the increasing satisfiability of node profiles, the decreasing error rate of IR-synopses tree and the decreasing query advantage of *tp-skq*, which leads to the smaller and smaller gap.

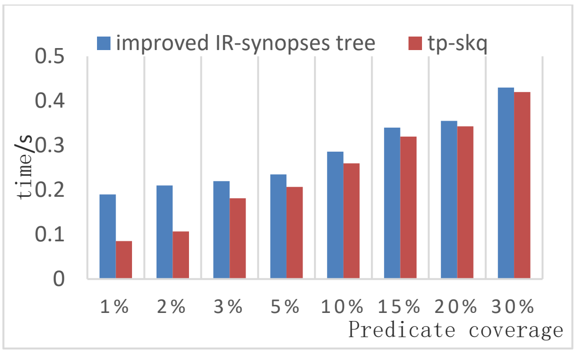
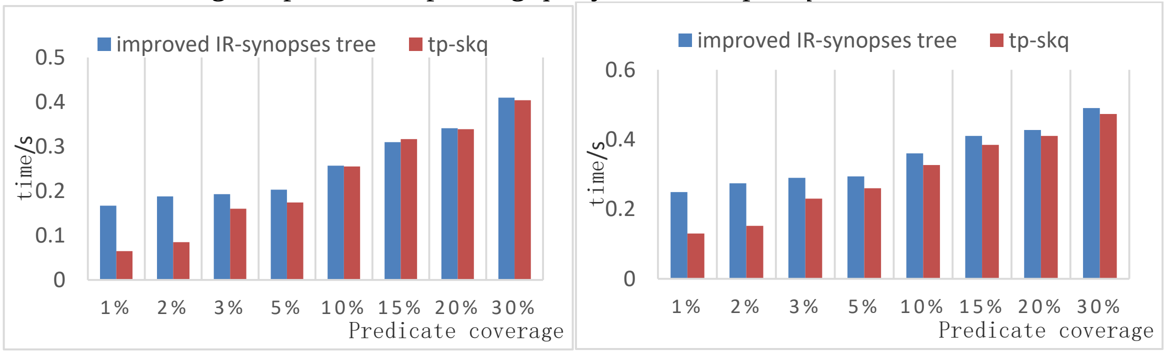


Figure 10. Predicate coverage change under 300000 attribute data set

In real life, users tend to be interested in objects with low coverage of predicates when querying predicates. For example, for restaurants with taste>9 and restaurants with taste score > 8, users often prefer the former, so it can be concluded that preference pruning queries based on *tp-skq* have better query performance.

When the amount of attribute data is 100,000 and 600,000, the query time of two different index structures under different predicate coverage is shown in Fig.11. From Fig.11, we can see that under 100,000 and 600,000 attribute data, the change rule of query time with the increase of predicate coverage is the same as that under 300,000 attribute data. With the increase of attribute data, the query time gap between the two index structures becomes more and more obvious. The query performance advantage of preference pruning query based on *tp-skq* is more and more obvious.



(a) (b)

Figure 11. Predicate coverage change under 100000 and 600000 attribute data set

5. Conclusion

Location-aware ranking query has become a research hotspot in recent years, but the research of location-aware ranking query based on user preference constraints has just begun. In this paper, a hybrid index structure of *tp-skq* is proposed to deal with location-aware ranking queries based on user preference constraints. A general pruning query algorithm based on *tp-skq* is given, and compared with existing algorithms in pruning rate, query time, index establishment time and so on. The experimental results show that the proposed index and its corresponding algorithm have the advantages of short index establishment time, high efficiency of pruning using user preference attributes and high query efficiency. Although the method proposed in this paper has obvious advantages, it still has some limitations. Therefore, the next step is to design an experimental system that can automatically select the optimal number of barrels in the global histogram according to the size of different attribute datasets when constructing *tp-skq*.

Acknowledgments

I would like to express my gratitude to all those who helped me during the writing of this thesis. First of all, I gratefully acknowledge the help of my supervisor, who has offered me valuable suggestions in the academic studies. Without her patient instruction, insightful criticism and expert guidance, the completion of this thesis would not have been possible. Second, I would like to sincerely thank the other students in the laboratory for their help and valuable suggestions in my study and life. Last, I should finally like to express my gratitude to my beloved parents who have always been helping me out of difficulties and supporting without a word of complaint.

References

[1] Liu XP, Wan CX, Liu DX, Liao GQ. Survey on spatial keyword search. Ruan Jian Xue Bao/Journal of Software, 2016, 27 (2):329−347 (in Chinese). http://www.jos.org.cn/1000-9825/4934.htm

[2] Chen L, GAO Y, Chen G, et al. Metric All-k-Nearest-Neighbor Search [J]. Knowledge & Data Engineering IEEE Transactions on, 2016, 28 (1): 1 - 1.

[3] Chen L, Cong G, Jensen C S, et al. Spatial Keyword Query Processing: An Experimental Evaluation [J]. Proceedings of the Vldb Endowment, 2013, 6 (3): 217 - 228.

[4] X. Liu, L. Chen, and C. Wan. LINQ: A Framework for Location-aware Indexing and Query Processing [J]. IEEE TKDE, 2015, 27 (5): 1288 - 1300.

[5] Li Z, Lee K C K, Zheng B, et al. IR-Tree: An Efficient Index for Geographic Document Search [J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23 (4): 585 - 599.

[6] Zhou AY, Yang B, Jin C, Ma Q. Location-Based services: Architecture and progress. Chinese Journal of Computers, 2011, 34 (7): 1155−1171 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01155].

[7] Chen L, Cong G, Jensen C S, et al. Spatial keyword query processing: an experimental evaluation[C]// International Conference on Very Large Data Bases. 2013.

[8] Zhang D, Tan K L, Tung A K H. Scalable top-k spatial keyword search[C]// International Conference on Extending Database Technology. 2013.

[9] Wu DM, Yiu ML, Cong G, Jensen CS. Joint top-K spatial keyword query processing. IEEE Trans. on Knowledge and Data Engineering, 2012, 24 (10): 1889−1903. [doi: 10.1109/TKDE.2011.172].

[10] Zhang D, Chee Y M, Mondal A, et al. Keyword Search in Spatial Databases: Towards Searching by Documen t [C]// IEEE International Conference on Data Engineering. 2009.

[11] U. Buranasaksee, K. Porkaew, "Multi-IRS: Multiple Trees Indexing for Generic Location-Aware Rank Query [C]", WCSE, 2016: 518 - 524.

[12] Buranasaksee U. Optimization of Textual Attribute Support in Generic Location-aware Rank Query[C]// IEEE International Conference on Computer and Communications. IEEE, 2017: 1206-1210.

[13] Tzoumas K, Deshpande A, Jensen C S. Lightweight Graphical Models for Selectivity Estimation without Independence Assumptions [J]. Pvldb, 2011, 4: 852 - 863.