International Journal of Database Theory and Application
Vol.9, No.3 (2016), pp. 181-190
http://dx.doi.org/10.14257/ijdta.2016.9.3.18

Top-k Algorithm for User Preferences based on Selection Strategy

Song Jin-ling [1], Liu Guo-hua[2], Liu Hai-bin[1], Huang Li-ming[1] andWu Yun-long[2]

[1] Hebei Normal University of Science & Technology, Qinhuangdao 066004, China
[2] College of Computer Science and Technology, Donghua University, Shanghai 201620, China
E-mail: Songjinling99@126.com

Abstract

In order to deal with multiple user preferences and improve query efficiency, selection strategy is adopted for top-k query to depress the compare operations. Firstly, the kth order statistics are selected randomly along with partitioning the data set basing on it, and the top-k result set can be received after several recursive partitions. Secondly, to select the kth order statistics accurately, the approximate kth order statistics is choose as threshold according to the similarity of user preference and system preference, and the top-k query result set can be accessed through simple comparison. Finally, the time complexities of presented algorithms are analyzed and their correctness and completeness are proved respectively. The experimental results show that our algorithms improve the efficiency of top-k query greatly.

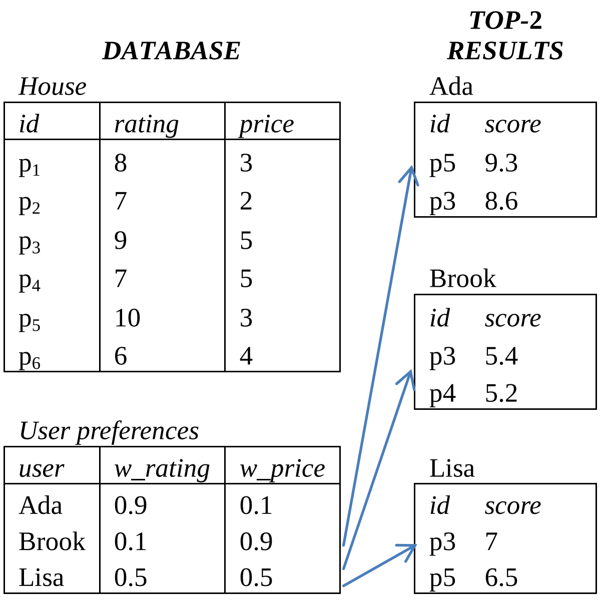
Keywords: *top-k query, user preferences, random selection, approximate selection*

1. Introduction

For an object table with scores or grades on each attribute and a defined function that combines the individual grade to an overall score, the top-*k* query on the table retrieves the only *k* objects with the highest scores about the function. Previous algorithms have focused on top-*k* query [1-13], one optimal algorithm was *TA* [4] which can stop scanning the object table quickly and differed from *FA* algorithm [10]. Basing on *TA* algorithm, many centralized or distributed top-*k* algorithms [5-9] were proposed. In addition, a partitioned layer-based index [11] was proposed for top-*k* algorithm, an interactive top-*k* algorithm was presented for spatial keyword queries [12], a top-*k* algorithm to select best represent objects was proposed on graph databases [13]. However, previous studies didn't focused on user preferences in top-*k* query, and they can only deal with single function not suitable for arbitrary functions. The concept of top-*k* query for user preferences is presented in Refs. [14-18], but what they are focusing is reverse top-*k* query not top-*k* query. So it is significant to study the top-*k* query based on user preferences.

For the top-*k* query based on user preferences, the function to combine scores is a linear function where weights are assigned to each scoring attribute indicating the importance of each attribute to the user. Because each individual pay diverse attention to each factor, namely different user have different preferences, so the top-*k* query must be scored according to the current user's preference. Consider a database containing information about different houses as well as user preferences, which is shown in Figure 1. For each house the *rating* and *price* are recorded and maximum value on each attribute is preferable. For each user the preference is denoted by different weights on each attribute, and different user may have different preference to a potential house, for instance, Ada is prefer the houses with high rating, but Brook is interested in houses with high price, Lisa equates between rating and price of houses. On the right part of Figure 1, the top-2 result set is depicted for Ada, Brook and Lisa respectively.

ISSN: 2005-4270 IJDTA



DATABASE

House

id	*rating*	*price*
p1	8	3
p2	7	2
p3	9	5
p4	7	5
p5	10	3
p6	6	4

User preferences

user	*w_rating*	*w_price*
Ada	0.9	0.1
Brook	0.1	0.9
Lisa	0.5	0.5

TOP-2 RESULTS

Ada

id	*score*
p5	9.3
p3	8.6

Brook

id	*score*
p3	5.4
p4	5.2

Lisa

id	*score*
p3	7
p5	6.5

Figure 1. An Instance of Top-*k* Query

For a table containing *N* objects (tuples), a naive top-*k* query algorithm based on user preferences is as following. Calculate scores of the front *k* objects firstly and save as the temporary top-*k* result set, then scan each of the rest *N*-*k* objects sequentially: if the object with minimal score in temporary result set is lower than the current object, then replace it with the current object. So the objects in the temporary result set are the top-*k* objects until the scan is finished. However, the time complexity of the algorithm is O (*k***N*), and it will be converted to O (N^2) if *k*=O (*N*), so the efficiency of the naive algorithm is low when *N* is considerably big. The low efficiency of naive algorithm is caused by a large number of compare operations, so the top-*k* query can be improved by reducing comparative times. In this paper we adopt selection strategy to operate top-*k* query. Firstly, we select the *k*th order statistics randomly and propose a top-*k* query algorithm. Secondly, we choose the approximate *k*th order statistics according to the similarity of user preference and system preferences, then top-*k* result set is received through simple comparison. The time complexities of these algorithms are analyzed and their correctness and completeness are proved. Experimental results show that our algorithms excel the natural algorithm with higher efficiency.

2. Definition

Let $S[U], U = \{A_1, A_2, ..., A_n\}$ be a relation schema, where A_1 is an integer column identifying each tuple (*i.e*. primary key), $A_i (i = 2,3,...,n)$ are real columns representing score. Relation S denotes the data set on $S[U]$. A tuple $p = < p[1],..., p[n] >$ in S expresses a data object, where $p[i] \in [0,10] (i = 2,3,...,n)$. Let $W[T], T = \{B_1, B_2, ..., B_n\}$ be a relation schema about user preferences, where B_1 is an integer column identifying each tuple (*i.e*. primary key), $B_i (i = 2,3,...,n)$ are real columns representing weights. Relation W denotes the data set on $W[T]$. A tuple $w = < w[1], w[2],..., w[n-1] >$ in W expresses a user preference, where $w[i] \in [0,1] (i = 2,3,...,n)$, $\sum_{i=2}^{n} w[i] = 1$.

Definition 1(Scoring Function [12]) The scoring function about tuple *p* and *w* is $f(w, p) = \sum_{i=2}^{n} (w[i] \cdot p[i])$, where $w \in W, p \in S$.

Definition 2 (Top-*k* Query [12]) Given a integer *k* and a user preference *w*, $TOP_k(w)$ is the result set of the top-*k* query if it satisfies: $TOP_k(w) \subseteq S, |TOP_k(w)| = k$ and for $\forall p_i, p_j : p_i \in TOP_k(w)$, $p_j \in (S - TOP_k(w))$,there is $f(w, p_i) \geq f(w, p_j)$.

When one or more objects have same score with the *k*th object, then any object of them can be returned.

Definition 3 (Data Dimensions) To a relation schema $S[U], U = \{A_1, A_2, ..., A_n\}$, since the attribute A_1 is only used as identifying attribute to support random access and improve query efficiency, A_1 is not a data dimension, so the data dimensions of $S[U]$ is $n-1$.

Definition 4 (User Preferences Similarity) to user preferences w_1, w_2 on relation W , the similarity $sim(w_1, w_2)$ of w_1, w_2 is denoted as:

$$sim(w_1, w_2) = (\sum_{i=2}^{n} (w_1[i] \cdot w_2[i]))^2 / (\sum_{i=2}^{n} (w_1[i])^2 \cdot \sum_{i=2}^{n} (w_2[i])^2) .$$

The symbols used in following algorithms are:

$result_{index=q}$: A tuple which value is *q* on *index* attribute in tuple set *result*;

$result_{index=q}[2]$: The second column value in tuple of $result_{index=q}$.

3. Top-*K* Algorithm Based on Random

The basic idea of top-*k* algorithm *RSTA* based on random selection: Firstly, the score on user preference of each object in data set *S* is counted to gain the materialized view *result*. Then the *RS-Partition* algorithm is called to divide the tuples in *result* repeatedly based on random selection strategy until the front partition contains *k* tuples. Finally the front *k* tuples are top-*k* query result set. *RSTA* algorithm is described as Algorithm 1.

```
Algorithm 1: RSTA (w, k, S)
Input: user preference w, constant k, data set S;
Output: The result set TK of top-k query on S;
Variable: temp is a temporary tuple.
index=0;
for each(p∈S) do
    temp[1]←p[1];
    temp[2]←f(w,p);
    temp[3]←index+1;
    result←result∪temp;
endfor
n←|result|;
RS-Partition (result,k,0,n);
TK←TK∪{ the front k tuples in result };
return TK;
```

The main procedures of the *RS-Partition* algorithm: Firstly, choose the *k*th tuple *r* in *result* as the *k*th score. Then partition the tuples in *result* according to the tuple *r*: the tuples whose score greater than *r* are moved to in front of it, the tuples whose score less than *r* are moved to the behind of it. If the tuples before *r* is more than *k* then call *RS-Partition* algorithm recursively on the former tuple set of *r*. If the tuples before *r* is less than *k* then call *RS-Partition* algorithm recursively on the behind tuple set of *r*. *RS-Partition* algorithm is described as Algorithm 2.

```
Algorithm 2: RS-Partition (result, k, p, q)
Input: tuple set result, constant k, starting location p, end location q;
Output: The partition data set basing on random statistic.
```

```
$r=result_{index=k}$;
$score \leftarrow r[2]$;
Exchange the values on front two columns of $r$ and $result_{index=q}$;
$score = result_{index=q}[2]$;
$i=p-1$;
for $j=p$ to $q-1$ /* Partition tuple set according to the score of $r$*/
  if $result_{index=j}[2] > score$ then
    $i=i+1$;
    Exchange the values on front two columns of $result_{index=i}$ and $result_{index=j}$;
  endif
endfor
Exchange the values on front two columns of $result_{index=i+1}$ and $result_{index=q}$;
$d=i+1$;
if($d>k$) then /* If the tuples before $r$ is more than $k$, then call RS-Partition recursively on the former tuples*/
   RS-Partition ($result,k,0,d$);
endif
if($d<k$) then /*If the tuples before $r$ is less than $k$, then call RS-Partition recursively on the behind tuples*/
   RS-Partition ($result,k-d,d+1,q$);
endif
Return;
```

The "for" circulation in *RS-Partition* algorithm partitions tuples in *result* in place, so it doesn't tie extra system space. According to the analysis of partition algorithm in Ref. [19], the average time complexity of *RS-Partition* algorithm is $O(N)$ when there are N tuples in S. If the tuple r is improving selected to make its score relatively intermediate, such as dividing the tuples in *result* to $|result|/5$ groups and choose the tuple which sore is the median of the median score in each group as r, then the worst case time complexity of *RS-Partition* will become $O(N)$.

4. Top-*K* Algorithm Based on Approximate Selection

The basic idea of top-k algorithm based on approximate selection: Firstly, algorithm *PA* preprocesses and generates a number of system preferences according to the data dimensions of data set S. Then the algorithm *PVA* preprocesses the tuples in data set S to be sorted by score according to each system preferences in W'. Finally, in the *ASTA* algorithm, we first search a system preference w_j which is most similar to the user preference w, after that we select the approximate kth order statistics of user preference w in the materialized view corresponding to w_j, and we get top-k query result set of S through simple comparison.

PA algorithm generates specified number of system preferences according to the data dimensions of data set S and saves them in set W'. *PA* algorithm is described as Algorithm 3.

```
Algorithm 3: PA($n$, $cnt$)
Input: data dimensions $n$, count of system preferences $cnt$;
Output: system preference set $W'$;
Variable: $w$ is a user preference, $temp$ is a system weight.
$W' \leftarrow \varnothing$;
$id \leftarrow 1$;
while($id \leq cnt$) do /* The circulation controls the number of system preferences */
  $w[1] \leftarrow id$;
  $w[2] \leftarrow id/cnt$;  /*The system weight on second dimension*/
```

```
    temp←(1-id/cnt)/(n-2); /* The system weights on other dimensions*/
    for(i←3;i≤ n;i←i+1) do /* Fill in system weights on other dimensions*/
      w[i]←temp;
    endfor
    W'←W'∪{w};
    id←id+1;
  endwhile
  return W';
```

The main steps of the *PVA* algorithm: Calculate the score of each tuple in data set *S* and join a temporary set *result* firstly. Then sort tuples in the *result* basing on score and add index value. Finally insert set *result* into the materialized view V_i which is united into the set *H*. *PVA* algorithm is described as Algorithm 4.

Algorithm 4: *PVA*(*W'*, *cnt*, *S*, *n*)

Input: system preference set $W'=\{w_1,w_2,\cdots,w_{cnt}\}$,the tuple count *cnt* of *W'*, data set *S*, data dimensions *n*;

Output: materialized view set *H*;

Variable: *temp* is a temporary tuple, *result* is the temporary tuple set, V_i is a materialized view.

```
  H←∅;
  for(i←1;i≤cnt;i←i+1) do  /* Generate a materialized view according to each system preference in W'*/
    result←∅;
    for each(p∈S) do  /*Calculate score to each tuple in S and save in the set result*/
      temp[1]←p[1];
      temp[2]←f(W'_{id=i}, p);
      result←result∪temp;
    endfor
    Descending sort the tuples in result basing on result[2];
    index←1;
    for each (p∈result) do  /*Adding index value to the sorted tuples */
      p[3]←index;
      index←index+1;
    endfor
    V_i←result;
    H←H∪{V_i};
  endfor
  return H;
```

The main steps of the *ASTA* algorithm: Firstly, search the system preference w_j which is most similar to the user preference *w*. Then, calculate the sore of the tuple corresponding to the *k*th tuple in the materialized view V_j in data set *S* and take it as the approximate *k*th order statistics of the top-*k* query, *i.e.* the query threshold. Finally, filter the tuples in data set *S* whose score are no less than the threshold and join *tempTK*, if the cardinality of *tempTK* is less than *k*, then append the remaining number of tuples with maximum score from *S*, if the cardinality of *tempTK* is greater than *k*, then delete redundant tuples with minimum score from *tempTK,* so the remaining tuples in *tempTK* are the result set of top-*k* query. ASTA algorithm is described as Algorithm 5.

Algorithm 5: *ASTA*(*w*, *k*, *S*, *W'*, *H*)

```
Input: user preference $w$, constant $k$, data set $S$, system preference set $W'=\{w_1,w_2,\ldots,w_{cnt}\}$, materialized view set $H=\{V_1,V_2,\ldots,V_{cnt}\}$ mapped to each system preference in $W'$;
Output: The result set $TK$ of top-$k$ query;
Variable: $tempTK$ stores the temporary result set.
$maxsim \leftarrow 0$;
$j \leftarrow 0$;
for($i \leftarrow 1; i \leq cnt; i \leftarrow i+1$) do  /*Search the system preference $w_j$ which is most similar to the user preference $w$ */
    if($sim(w,w_i) > maxsim$)
        maxsim$\leftarrow sim(w,w_i)$;
        $j \leftarrow i$;
    endif
endfor
$r \leftarrow (V_j)_{index=k}$; /*Save the $k$th tuples in the materialized view $V_j$ corresponding to $w_j$ */
$score \leftarrow f(w, S_{id=r[1]})$; /*Calculate the sore of the corresponding tuple in data set $S$ and look it as the approximate query threshold*/
$count \leftarrow 0$;
$tempTK \leftarrow \Phi$;
for each($p \in S$) do /*Calculate the score of each tuple in data set $S$ and filter the tuples whose score are no less than threshold */
    $temp[1] \leftarrow p[1]$;
    $temp[2] \leftarrow f(w, p)$;
    if($temp[2] >= score$)
        $tempTK \leftarrow tempTK \cup \{temp\}$;
        $count \leftarrow count+1$;
    else
        $result \leftarrow result \cup temp$;
    endif
endfor
if ($count>k$) then /*Delete ($count$-$k$) tuples with minimum score from $tempTK$ */
    $d \leftarrow count-k$;
    while($d>0$) do
        $r \leftarrow$The tuple with minimal score in $tempTK$;
        $tempTK \leftarrow tempTK-\{r\}$;
        $d \leftarrow d-1$;
    endwhile
endif
if ($count<k$) then /*Append ($k$-$count$) tuples with maximal score in $result$ to $tempTK$ */
    $d \leftarrow k-count$;
    while($d>0$) do
        $r \leftarrow$The tuple with maximal score in $result$;
        $tempTK \leftarrow tempTK \cup \{r\}$;
        $result \leftarrow result-\{r\}$;
        $d \leftarrow d-1$;
    endwhile
endif
return $TK$;
```

The time complexity of the *ASTA* algorithm is determined by 4 circles, if there are N tuples in the data set S, the time complexity for each circle is analyzed as following:

(1) The worst-case time complexity of the first circle is $O(cnt)$;

(2) The worst-case time complexity of the second circle is $O(N)$;

(3) The time complexity of third circle: Since the circle loops ($count$-k) times, the statement " Select the tuple with minimal score in *tempTK* " executes (N-$count$) times at most, then the time complexity of the circle is O (($count$-k) (N-$count$)) which can turn to O (C_1N- C_1count) if let C_1= $count$-k, so the worst-case time complexity of it is O (N);

(4) The time complexity of fourth circle: Since the circle can loop (k-$count$) times, the statement " Select the tuple with maximal score in *result* " executes (N-$count$) times at most, then the time complexity of the circle is O ((k-$count$) (N-$count$)) which can turn to O (C_2N- C_2count) if let C_2= k-$count$, so the worst-case time complexity of it is O (N);

Overlying the time complexity of above 4 circles, the time complexity of *ASTA* is O (N+cnt), because the cnt is very small relative to N, so the worst-case time complexity of *ASTA* is O (N).

5. The Correctness and Completeness of Algorithms

Theorem 1 *RSTA* and *ASTA* algorithm can find the accurate top-k objects.

Proof: In *RSTA* algorithm a random selected tuple is used to partition data set S: tuples which score are greater than it moved to the front, and other tuples moved to the back. The partition will be repeated until there are k tuples in front of the selected tuple. Because the front k tuples are all larger than behind tuples, so they are the right top-k objects.

In *ASTA* algorithm sub query result *tempTK* is obtained based on the approximate threshold. So *tempTK* is a top-n result on the data set S while n is not certain, *i.e.*, *tempTK* is the largest n tuples sorted by the current user preference. However the third and the fourth loops in the algorithm can guarantee that n=k, so the final top-k objects is certainly accurate.

Theorem 2 *RSTA* and *ASTA* algorithm can find all the accurate top-k objects.

Proof: In *RSTA* algorithm the tuples are divided directly on the original data set basing on score, which can ensure the score of former k tuples are higher than other n-k tuples, so *RSTA* algorithm can find all of the top-k objects. By Theorem 1, *RSTA* algorithm can find the right top-k objects, so *ASTA* algorithm can find all the accurate top-k objects.

In *ASTA* algorithm objects with higher score are joined *tempTK* based on the approximate threshold. To ensure *tempTK* containing k objects with largest score, redundancy objects with smaller score are deleted if the cardinality of *tempTK* is more than k, objects with largest score in remainder tuples are appended if the cardinality of *tempTK* is less than k, so *ASTA* algorithm can find all the top-k objects. By Theorem 1, *ASTA* algorithm can find the accurate top-k objects. In conclusion *ASTA* algorithm can find all the accurate top-k objects.

6. Experimental

Our experiments mainly compare and analyze the efficiency of *RSTA*, *ASTA* and the natural algorithm (named *Naive*) to the variation of data dimensions, k value and tuples. The experimental environment is: win7, processor of Intel (R) Core (TM) 2 Duo CPU E8400 @ 3.00GHz (2CPUs), ~3.0GHz, DBMS of Microsoft SQL Server 2008 R2. Data set is generated by the data synthesizer of IBM.

6.1 Experimental Preparation

In database *HousesAndUsers*, tuples in *House* table are generated by the IBM data synthesizer firstly, then *Systempreference* table is created according to the data dimensions of *House* table to represent system preferences. The views generated basing on system preferences are stored in database *UserViews*. Tables *tempTK* and *TK* are created to store intermediate result and the final result respectively, where *tempTK* will be emptied before *RSTA* or *ASTA* algorithm are running to avoid the influence of the last result.

6.2 Results and Discussion

We compare the running time on *RSTA*, *ASTA* and *Naive* algorithm firstly with fixed tuples and *k* value but varying data dimensions. Without loss of generality, the tuples in *House* table is 20000, *k* value is 30, the data dimensions are 2, 4, 6, 8, 10, top-*k* query time of *RSTA*, *ASTA* and *Naive* algorithm are shown in Figure 2. As we can see from Figure 2, the running time of *RSTA*, *ASTA* and *Naive* algorithm changes a little along with the change of data dimensions, which indicates that *RSTA*, *ASTA* and *Naive* algorithm spent unconcerned with the data dimensions.

Then, we compare the running time on *RSTA*, *ASTA* and *Naive* algorithm with fixed tuples and data dimensions but varying *k* value. Without loss of generality, the tuples in *House* table is 20000, data dimensions are 10, *k* varies from 3, 5, 10, 20, 30, 50 respectively, top-*k* query time of *RSTA*, *ASTA* and *Naive* algorithm are shown in Figure 3. It is shown that the running time of *RSTA*, *ASTA* algorithm change slowly, but *Naive* algorithm suffers large change when *k* varies, which suggests that *RSTA* and *ASTA* are further insulated with *k* value, but the *Naive* algorithm is affected by *k* value obviously.

Finally, we compare the running time on *RSTA*, *ASTA* and *Naive* algorithm with fixed *k* value and data dimensions but varying tuples. Without loss of generality, the data dimensions are 10, *k* value is 30, and tuples in *House* table change from 5000 to 50000, top-*k* query time of *RSTA*, *ASTA* and *Naive* algorithm are shown in Figure 4. We know from Figure 4 that *RSTA* and *ASTA* algorithm are running faster than *Naive* algorithm, and their spending varies smoothly than *Naive* algorithm with the increase of tuples, which show that *Naive* algorithm is influenced by tuples greatly than *RSTA* and *ASTA* algorithm.

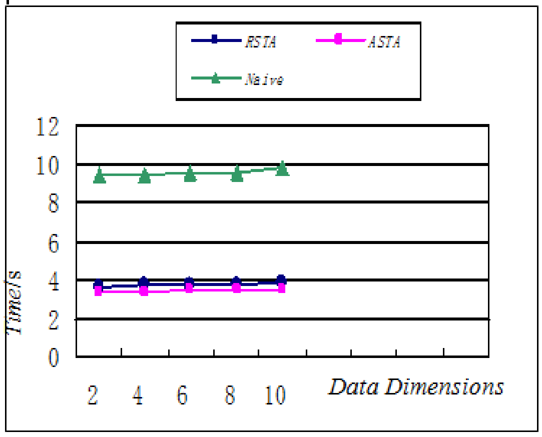


Figure 2. Comparison of Query Time with Varying Data Dimensions

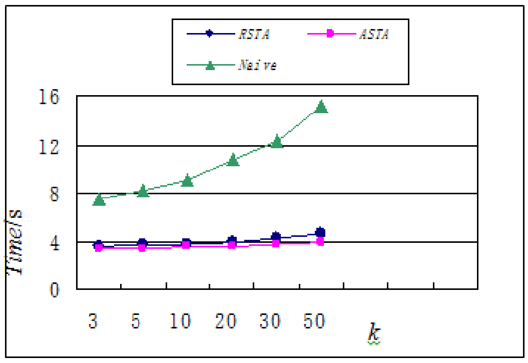


Figure 3. Comparison of Query Time with Varying *K* Values

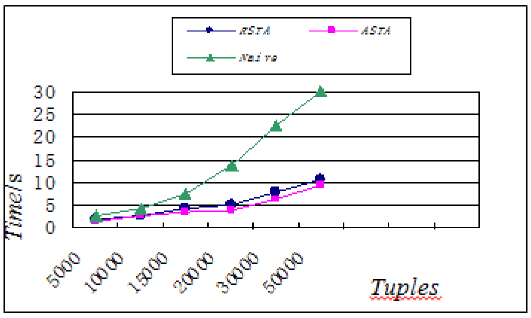


Figure 4. Comparison of Query Time with Varying Tuples

In summary, *RSTA* and *ASTA* algorithm are more efficient comparing to *Naive* algorithm, so *RSTA* and *ASTA* algorithm improve the response of the top-*k* query well.

7. Conclusion

Top-*k* query based on user preferences can deal with query problem on multi-user preferences and various *k* values. *RSTA* algorithm produces top-*k* result set by random selected order statistics. *ASTA* algorithm generates top-*k* result set through approximate threshold. Further study will focus on top-*k* query algorithms with lower time complexity and suitable to dynamic data set.

Acknowledgement

This work is supported by The Natural Science youth Foundation of Hebei Province (No. F2015407039), Hebei Province Department of science and technology Fund (NO.13227427), Hebei Province Department of Education Fund (NO.sz132075, sz141097, 2012-112), Doctoral Foundation of HeBei Normal University of Science & Technology (2013YB007, 2013YB001).

References

[1] R. Akbarinia, E. Pacitti and P. Valduriez, "Best position algorithms for top-k queries", Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria, **(2007)**.
[2] Y. C. Chang, L. D. Bergman and V. Castelli, "The onion technique: indexing for linear optimization queries", Proceedings of SIGMOD 2000, Dallas, USA, **(2000)**.
[3] G. Das, D. Gunopulos and N. Koudas, "Answering top-k queries using views", Proceedings of the 32nd international conference on Very large data bases, Seoul, Korea, **(2006)**.
[4] R. Fagin, A. Lotem and M. Naor, "Optimal aggregation algorithms for middleware", Journal of Computer and System Sciences, vol. 4, no. 66, **(2003)**, pp. 614-656.
[5] S. Ge, N. Mamoulis and D. Cheung, "Efficient All Top-k Computation - A Unified Solution for All Top-

k, Reverse Top-k and Top-m Influential Queries", IEEE Transactions, vol. 5, no. 25, **(2013)**, pp. 1015-1027.

[6] I. F. Ilyas, W. G. Aref and A. K. Elmagarmid, "Supporting top-k join queries in relational databases", The VLDB Journal, vol. 3, no. 13, **(2004)**, pp. 207-221.

[7] I. F. Ilyas, G. Beskales and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems", ACM Computing Surveys, vol. 4, no. 40, **(2008)**, pp. 11.

[8] S. W. Hwang and K. C. C. Chang, "Optimizing top-k queries for middleware access: A unified cost-based approach", ACM Transactions on Database Systems, vol. 1, no. 32, **(2007)**, pp. 5.

[9] D. Xin, C. Chen and J. Han, "Towards robust indexing for ranked queries", Proceedings of the 32nd International conference on Very large data bases, Seoul, Korea, **(2006)**.

[10] R. Fagin, "Combining fuzzy information from multiple systems", Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Montreal, Canada, **(1996)**.

[11] S. Y. Ihma, K. E. Leeb, A. Nasridinovd, J. S. Heoc and Y. H. Park, "Approximate Convex Skyline: A Partitioned Layer-based Index for Efficient Processing Top-k Queries", Knowledge-Based Systems, vol. 5, no. 61, **(2014)**, pp. 13-28.

[12] K. Zheng, H. Su, B. Zheng, J. Xu, J. Liu and X. Zhou, "Interactive Top-k Spatial Keyword Queries", Proceedings of ICDE 2015, Seoul, Korea, **(2015)**.

[13] S. Ranu, M. M. Hoang and A. Singh, "Answering Top-k Representative Queries on Graph Databases", Proceedings of SIGMOD 2014, Snowbird, USA, **(2014)**.

[14] A. Arvanitis, A. Deligiannakis and Y. Vassiliou, "Efficient influence-based processing of market research queries", Proceedings of the 21st ACM international conference on Information and knowledge management, Maui, USA, **(2012)**.

[15] A. Vlachou, C. Doulkeridis and Y. Kotidis, "Reverse top-k queries", Proceedings of 26th International Conference on Data Engineering, Long Beach, USA, **(2010)**.

[16] A. Vlachou, C. Doulkeridis, K. Nørvåg, "Identifying the most influential data objects with reverse top-k queries", Proceedings of the VLDB Endowment, Singapore, **(2010)**.

[17] A. Vlachou, C. Doulkeridis and Y. Kotidis, "Monochromatic and bichromatic reverse top-k queries", IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 23, **(2011)**, pp. 1215-1229.

[18] A. Vlachou, C. Doulkeridis and K. Nørvåg, "Branch-and-Bound Algorithm for Reverse Top-k Queries", Proceedings of the SIGMOD 2013, New York, USA, **(2013)**.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, P. Jingui, G. Tiecheng, L. Chengfa and Y. Mao, "Introduction to Algorithms", China Mechine Press, Beijing, **(2006)**.