Top-k Algorithm for User Preferences based on Selection Strategy

Song Jin-ling¹, Liu Guo-hua², Liu Hai-bin¹, Huang Li-ming¹ and Wu Yun-long²

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

Definition 1(Scoring Function [12]) The scoring function about tuple \( p \) and \( w \) is 
\[
f(w,p) = \sum_{i=2}^{n} (w[i] \cdot p[i]), \quad \text{where} \quad w \in W, \quad p \in S.
\]
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 $\forall p, p_j : p \in TOP_k(w), p_j \in (S - TOP_k(w))$, there is $f(w, p) \geq f(w, p_j)$.

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 $S(U) = \{A_1, A_2, \ldots, A_n\}$, since the attribute $A_i$ is only used as identifying attribute to support random access and improve query efficiency, $A_i$ is not a data dimension, so the data dimensions of $S(U)$ is $n - 1$.

Definition 4 (User Preferences Similarity) To user preferences $w_i, w_j$ on relation $w$, the similarity $\text{sim}(w_i, w_j)$ of $w_i, w_j$ is denoted as:

$$\text{sim}(w_i, w_j) = \frac{\sum_{i=2}^{n} (w_i[i] \cdot w_j[i])}{\sqrt{\sum_{i=2}^{n} (w_i[i])^2 \cdot \sum_{i=2}^{n} (w_j[i])^2}}.$$

The symbols used in following algorithms are:

- $\text{result}_{\text{index}=q}[1]$ : The second column value in tuple of $\text{result}_{\text{index}=q}$.

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 $\text{result}$. Then the $R$-Partition algorithm is called to divide the tuples in $\text{result}$ 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: $\text{temp}$ is a temporary tuple.

$\text{index} = 0$;

for each ($p \in S$) do

$\text{temp}[1] \leftarrow p[1]$;

$\text{temp}[2] \leftarrow f(w, p)$;

$\text{temp}[3] \leftarrow \text{index} + 1$;

$\text{result} \leftarrow \text{result} \cup \text{temp}$;

endfor

$n \leftarrow |\text{result}|$;

$\text{RS-Partition (result, k, 0, n)}$;

$\text{TK} \leftarrow \text{TK} \cup \{ \text{the front } k \text{ tuples in result} \}$;

return $\text{TK}$;

The main procedures of the $\text{RS-Partition}$ algorithm: Firstly, choose the $k$th tuple $r$ in $\text{result}$ as the $k$th score. Then partition the tuples in $\text{result}$ 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 $\text{RS-Partition}$ algorithm recursively on the former tuple set of $r$. If the tuples before $r$ is less than $k$ then call $\text{RS-Partition}$ algorithm recursively on the behind tuple set of $r$. $\text{RS-Partition}$ algorithm is described as Algorithm 2.

**Algorithm 2:** $\text{RS-Partition (result, k, p, q)}$

Input: tuple set $\text{result}$, constant $k$, starting location $p$, end location $q$;

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

The “for” circulation in \( RS-\text{Partition} \) algorithm partitions tuples in \( \text{result} \) 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 \( RS-\text{Partition} \) 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 \( \text{result} \) to \( |\text{result}|/5 \) 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 \( RS-\text{Partition} \) 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 \( k \)th 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.
\[ W^* \leftarrow \emptyset; \]
\[ id \leftarrow 1; \]
\[ \text{while} (id \leq cnt) \text{ do } /* \text{The circulation controls the number of system preferences} */ \]
\[ w[1] \leftarrow id; \]
\[ w[2] \leftarrow id/cnt; /* \text{The system weight on second dimension} */ \]
temp←(1-id\textit{cnt})/(n-2); /* The system weights on other dimensions*/
\[\text{for}(i←3;i≤n;i←i+1) \text{ do }/* Fill in system weights on other dimensions*/\]
\[w[i]←\text{temp};\]
endfor

\[W'←W∪\{w\};\]
\[id←id+1;\]
endwhile

return \(W'\);

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

\textbf{Algorithm 4:} \textit{PVA}(W', \text{cnt}, S, n)

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

Output: materialized view set \(H\);

Variable: \(\text{temp}\) is a temporary tuple, \(\text{result}\) is the temporary tuple set, \(V_i\) is a materialized view.

\[H←\emptyset;\]
\[\text{for}(i←1;i≤\text{cnt};i←i+1) \text{ do }/* Generate a materialized view according to each system preference in } W'*/\]
\[\text{result}←\emptyset;\]
\[\text{for each}(p∈S) \text{ do }/*\text{Calculate score to each tuple in } S\text{ and save in the set } \text{result}*/\]
\[\text{temp[1]}←p[1];\]
\[\text{temp[2]}←f(W'_{\text{id\textit{cnt}}}, p);\]
\[\text{result}←\text{result}∪\text{temp};\]
endfor

Descending sort the tuples in \(\text{result}\) basing on \(\text{result}[2]\);
\[\text{index}←1;\]
\[\text{for each} (p∈\text{result}) \text{ do }/*\text{Adding index value to the sorted tuples */}\]
\[p[3]←\text{index};\]
\[\text{index}←\text{index}+1;\]
endfor

\[V_i←\text{result};\]
\[H←H∪\{V_i\};\]
endfor

return \(H\);

The main steps of the \textit{ASTA} algorithm: Firstly, search the system preference \(w_j\) which is most similar to the user preference \(w\). Then, calculate the score 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, \textit{i.e.} the query threshold. Finally, filter the tuples in data set \(S\) whose score are no less than the threshold and join \(\text{tempTK}\), if the cardinality of \(\text{tempTK}\) is less than \(k\), then append the remaining number of tuples with maximum score from \(S\), if the cardinality of \(\text{tempTK}\) is greater than \(k\), then delete redundant tuples with minimum score from \(\text{tempTK}\), so the remaining tuples in \(\text{tempTK}\) are the result set of top-\(k\) query. \textit{ASTA} algorithm is described as Algorithm 5.

\textbf{Algorithm 5:} \textit{ASTA}(w, k, S, W', H)
Input: user preference \( w \), constant \( k \), data set \( S \), system preference set 
\( W = \{w_1, w_2, \ldots, w_{\text{cnt}}\} \), materialized view set 
\( H = \{V_1, V_2, \ldots, V_{\text{cnt}}\} \) mapped to each system
preference in \( W \);

Output: The result set \( TK \) of top-\( k \) query;

Variable: \( tempTK \) stores the temporary result set.

\[
\text{maxsim} \leftarrow 0; \quad j \leftarrow 0; \\
\text{for} (i \leftarrow 1; i \leq \text{cnt}; i \leftarrow i+1) \quad /*\text{Search the system preference } w_j \text{ which is most} \\
\text{sim} \leftarrow \text{sim}(w, w_i) \quad /*\text{similar to the user preference } w */ \\
\text{maxsim} \leftarrow \text{maxsim}(w, w_i); \\
\text{endif} \\
\text{endfor} \\
\text{r} \leftarrow (V_j)_{\text{index}=k}; /*\text{Save the } k \text{th tuples in the materialized view } V_j \text{ corresponding to } w_j */ \\
\text{score} \leftarrow f(w, S_{\text{id}=1}); /*\text{Calculate the sore of the corresponding tuple in data set } S \text{ and} \\
\text{look it as the approximate query threshold} */ \\
\text{count} \leftarrow 0; \\
\text{tempTK} \leftarrow \Phi; \\
\text{for each} (p \in S) \quad /*\text{Calculate the score of each tuple in data set } S \text{ and filter} \\
\text{the tuples whose score are no less than threshold} */ \\
\text{temp}[1] \leftarrow p[1]; \quad \text{temp}[2] \leftarrow f(w, p); \quad /*\text{Calculate the score of each tuple in data set } S \text{ and filter} \\
\text{the tuples whose score are no less than threshold} */ \\
\text{if} (\text{temp}[2] \geq \text{score}) \\
\text{tempTK} \leftarrow \text{tempTK} \cup \{\text{temp}\}; \quad \text{count} \leftarrow \text{count}+1; \\
\text{else} \\
\text{result} \leftarrow \text{result} \cup \text{temp}; \\
\text{endif} \\
\text{endfor} \\
\text{if} (\text{count} > k) \quad /*\text{Delete (count-}k \text{) tuples with minimum score from } tempTK */ \\
\text{d} \leftarrow \text{count}-k; \\
\text{while} (\text{d}>0) \text{ do} \\
\text{r} \leftarrow \text{The tuple with minimal score in } tempTK; \quad /*\text{Delete (count-}k \text{) tuples with minimum score from } tempTK */ \\
\text{tempTK} \leftarrow \text{tempTK} \setminus \{r\}; \quad \text{d} \leftarrow \text{d}-1; \\
\text{endwhile} \\
\text{endif} \\
\text{if} (\text{count} < k) \quad /*\text{Append (k-count) tuples with maximal score in } result \text{ to} \\
\text{tempTK} */ \\
\text{d} \leftarrow k-\text{count}; \\
\text{while} (\text{d}>0) \text{ do} \\
\text{r} \leftarrow \text{The tuple with maximal score in } result; \quad /*\text{Append (k-count) tuples with maximal score in } result \text{ to} \\
\text{tempTK} */ \\
\text{tempTK} \leftarrow \text{tempTK} \cup \{r\}; \quad \text{result} \leftarrow \text{result} \setminus \{r\}; \quad \text{d} \leftarrow \text{d}-1; \\
\text{endwhile} \\
\text{endif} \\
\text{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 \( \text{(count-} k \text{-)} \) times, the statement " Select the tuple with minimal score in tempTK " executes \( (N\text{-count}) \) times at most, then the time complexity of the circle is \( O((\text{count-} k \text{-)} (N\text{-count})) \) which can turn to \( O(CN- C\text{count}) \) if let \( C= \text{count-} k \), so the worst-case time complexity of it is \( O(N) \);
4. The time complexity of fourth circle: Since the circle can loop \( (k\text{-count}) \) times, the statement " Select the tuple with maximal score in result " executes \( (N\text{-count}) \) times at most, then the time complexity of the circle is \( O((k\text{-count}) (N\text{-count})) \) which can turn to \( O(CN- C\text{count}) \) if let \( C= k\text{-count} \), 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 base 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](image-url)
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

[5] S. Ge, N. Mamoulis and D. Cheung, “Efficient All Top-k Computation - A Unified Solution for All Top-