Efficient Top-K Query Algorithms Using Density Index

Dongqu Chen[1], Guang-Zhong Sun[1*] and Neil Zhenqiang Gong[1**]
[1]*Key Laboratory on High Performance Computing, Anhui Province*
School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, P.R. China
cdq2012@mail.ustc.edu.cn; gzsun@ustc.edu.cn; neilz.gong@berkeley.edu

***Abstract* - Top-*k* query has been widely studied recently in many applied fields. Fagin et al. [3] proposed an efficient algorithm, the Threshold Algorithm (i.e. TA), to process top-*k* queries. However, in many cases, TA does not terminate even if the final top-*k* results have been found for some time. Based on these, we propose a novel algorithm: Density Threshold Algorithm (i.e. DTA), which is designed to minimize the useless accesses of a top-*k* query, and introduce a novel indexing structure, *Density Index*, to support our algorithms. However, we proved the DTA is not *instance optimal* in Fagin's notion and we also propose an instance optimal algorithm named Selective-Density Threshold Algorithm (i.e. S-DTA). Finally, extensive experiments show that our algorithms have significant improvement on the efficiency, compared with the TA algorithm.**

Index Terms* - *Database query processing, Algorithms, Indexes.

I. Introduction

Ranking aware queries, or top-*k* queries, have been widely studied recently in many applied fields such as information retrieval, multimedia databases and data mining. The main reason for such attention is that top-*k* queries avoid overwhelming the user with large numbers of uninteresting answers which are resource-consuming.

A general and simple model proposed by Fagin et al. [3] is that the dataset consists of *m* sorted lists with *n* data items. Under this model, Fagin et al. [3] proposed the efficient Threshold Algorithm (i.e. TA). To measure the optimality of an algorithm, they defined a notion of optimality, *instance optimality* and proved the instance optimality of TA.

However, in many cases, TA does not terminate even if the final top-*k* results have been found for some time, which will bring more cost and delay to the queries. The main reason for such useless accesses is the *threshold value* is not low enough to satisfy the terminate condition of TA. To speed up the reduction of the threshold value, U. Güntzer et al. [4] used the reducing speed of the accessed items to predict the speed of the unknown items in the same list. They considered that it could speed up the reduction of the threshold value if the list with the largest predicted value was chosen to be accessed in every step. Our target is similar with U. Güntzer but we reduce the threshold value through choosing the *section* with the largest *lean value* and accessing the lists *section* by *section*, based on the *Density Index* set up in the pre-computing phase.

In this paper, we study the efficient top-*k* queries using pre-computed analysis and indexing method. We propose a novel algorithm: Density Threshold Algorithm (i.e. DTA), and we also turn DTA into an instance optimal algorithm named Selective-Density Threshold Algorithm (i.e. S-DTA). Finally, extensive experiments show that our algorithms have significant improvement on the efficiency, compared with the TA algorithm.

II. Computation Model And TA Algorithm

Our model of the dataset can be described as follows [3]: assume the database D consists of m sorted lists, which are denoted as $L_1, L_2 \ldots L_m$. Each sorted list consists of n data items. We may refer to L_i as list i. Each entry of L_i is of the form $(x, s_i(x))$ where x is an object and $s_i(x)$ is the *ith* local score of x as a positive real number in the interval [0, 1]. Sorted list means that objects in each list are sorted in descending order by the $s_i(x)$ value. For a given object x, x has a total score of $f(x)=f(s_1(x), s_2(x) \ldots s_m(x))$, where the m-dimensional aggregate function f is supposed to be increasingly monotonic:

Definition 2.1 **Aggregate Monotone Function** [3]. *An aggregate function f is monotone if* $f(a_1, a_2 \ldots a_m) \leq f(a_1', a_2' \ldots a_m')$, *whenever* $a_i \leq a_i'$ *for every i.*

In this paper, we assume the aggregate function is weighted summation function, yielding $f(x)=\sum_{i=1}^{m} w_i s_i(x)$ and $\sum_{i=1}^{m} w_i = 1$ ($w_i \neq 0$), the most common form of aggregate function in applications.

Each data item can be accessed through *sorted access* or *random access*. Sorted access iteratively reads data items sequentially, whereas a random access is a request for a data item in some list given the object's ID. The middleware cost of a top-*k* query algorithm is $a_S c_S + a_R c_R$, where a_S is the number of sorted accesses performed, a_R is the number of random accesses performed, c_S is the cost per sorted access, and c_R is the cost per random access.

Our task is to determine the top-*k* objects, that is, *k* objects with the highest total scores. To solve the top-*k* query described above, Fagin et al. [3] proposed the threshold algorithm (i.e. TA) as described in Fig. 1.

*Corresponding author
**Neil Z. Gong is now a postgraduate in EECS, UC Berkeley. This work was completed when Neil Z. Gong was undergraduate student of USTC.

Threshold Algorithm (TA)

1. Do sorted access in parallel to each of the m lists. As an object is seen through sorted access in some list, do random access to the other lists to find all its missing local scores, and compute its total score. Maintain a set Y containing the k objects whose total scores are the highest among all the objects seen so far.
2. For each list L_i, let be $\underline{s_i}$ the bottom score of L_i, which is the last local score seen under sorted access in L_i. Define the *threshold value* to be $\tau = f(\underline{s_1}, \underline{s_2} \ldots \underline{s_m})$.
3. Halt when $\tau \leq M_k$, where $M_k = \min\{ f(x) \mid x \in Y\}$.

Figure 1. Threshold Algorithm

III. Density Threshold Algorithm

Before proposing our algorithm, we first introduce an efficient indexing structure, Density Index to support the DTA.

Firstly, we divide each attribute list into b sections and the jth section Sn_j consists of the items whose local scores is in the associated score interval $(1 - j \times (1/b), 1 - (j - 1) \times (1/b)]$, where b is a positive integer and $j = 1, 2 \ldots b$. Items with score 0 belong to the bth section Sn_b.

Obviously, a section is a sorted sub-list of one attribute list.

Definition 3.1 **Density**. *For section Sn_j, let size be the number of items in Sn_j. If size is positive, the density of Sn_j is 1/size, denoted as d_j. Sections of zero size do not have density.*

Definition 3.2 **Head Item and Tail Item**. *For section Sn_j, head item is the first item in Sn_j while tail item is the last one.*

Obviously, the head item has the highest local score in the section whereas the tail item is of the lowest score.

Density Index is an indexing structure to remember the density information of every section in every list. It can be m indexing lists corresponding to m attribute lists. Sections of zero *size* are removed from the index.

Now we show the Density Threshold Algorithm in this section.

Definition 3.3 **Availability**. *An item in L_i is available if and only if all the items above it in L_i have been sorted or random accessed before while the item itself has not been sorted accessed before. Similarly we say a section is available if and only if one of its items is available.*

Definition 3.4 **Lean Value**. *For section Sn_j in the ith list and aggregation function $f(x) = \sum_{i=1}^{m} w_i s_i(x)$, the lean value of Sn_j is $w_i d_j$.*

Obviously, in order to speed up the reduction of the threshold value, we need to access the available section with the largest lean value. Based on this, we now propose the Density Threshold Algorithm (i.e., DTA), which is described by the pseudo-code in Fig. 2.

Theorem 3.1. *If the aggregation function f is monotone, then DTA correctly finds the top-k answers.*

Proof: Let set Y be the result set DTA returned. According to the algorithm, DTA halts if and only if $Y.size = k$ and $\tau \leq M_k$. Hence, for every object $y \in Y$ and every $z \notin Y$:

Density Threshold Algorithm (DTA)

Pre-computing Phase:

Build the Density Index of the given database.

Computing Phase:

1: $Y = \emptyset$, $\tau = 0$, $M_k = 0$, $\underline{s_i} = 1$ where $i = 1, 2 \ldots m$.
2: **while** ($Y.size < k$ or $\tau > M_k$) **do**
3: Let Sn_j be the section with the largest lean value in the available sections according to the Density Index (if the largest lean value is reached for more than one sections, any of them can be chosen randomly).
4: **for** item $(x, s_i(x))$ = **from** the head item **down to** the tail item of Sn_j **do**
5: $\underline{s_i} = s_i(x)$.
6: $\tau = f(\underline{s_1}, \underline{s_2} \ldots \underline{s_m})$.
7: **if** object x has not been accessed before **then**
8: get the missing local scores of the object x and calculate the total score $f(x)$;
9: **if** ($Y.size < k$) **then**
10: insert x into Y;
11: Let t be the object with the lowest total score in Y and M_k be its total score;
12: **else**
13: **if** ($f(x) > M_k$) **then**
14: remove t from Y and insert x into Y;
15: Let t be the object with the lowest total score in Y and M_k be its total score;
16: **if** ($\tau \leq M_k$) **then**
17: go to 20.
18: **end for.**
19: **end while.**
20: **Return** Y.

Figure 2. Density Threshold Algorithm

Case 1: If z has been seen before DTA halts and finally not in Y, the fact is that there exist at least $Y.size = k$ objects with higher total scores than z.

Case 2: If z has not been seen by DTA and assume that the local scores of z are $s_i(z)$ where $i = 1, 2 \ldots m$, then we have $s_i(z) \leq \underline{s_i}$. Therefore, $f(z) \leq f(\underline{s_1}, \underline{s_2} \ldots \underline{s_m}) = \tau \leq M_k$. Since M_k is the lowest total score in Y, every y has $f(y) \geq M_k \geq f(z)$.

Therefore, z has no chance to be one top-k answer in both cases, as desired. □

IV. Selective-Density Threshold Algorithm

To measure the optimality of an algorithm, Fagin et al. [3] defined a notion of optimality, *instance optimality* as following.

Definition 4.1 **Instance Optimality [3]**. *We say that an algorithm $\mathcal{B}$ is instance optimal over* **A** *and* **D** *if $\mathcal{B} \in$ **A** and if for every $\mathcal{A} \in$ **A** and every $\mathcal{D} \in$ **D** we have $cost(\mathcal{B}, \mathcal{D}) = O(cost(\mathcal{A}, \mathcal{D}))$, that is, there are constants c and c' satisfying $cost(\mathcal{B}, \mathcal{D}) \leq c \cdot cost(\mathcal{A}, \mathcal{D}) + c'$.*

We first show an example that demonstrates DTA is not instance optimal.

TABLE I. DATASET OF EXAMPLE 1

L_1	L_2
(R_1, 1.00)	(R_n,1.00)
(R_2, 0.95)	(R_{n-1},0.85)
..... (R_i , $0.9 + (n-1-i)\times \frac{0.05}{n-3}$)	..... (R_i , $0.75 + (i-2)\times \frac{0.10}{n-3}$) and ($R_1$, 0.80) .....
(R_{n-1},0.90)	
(R_n, 0.30)	(R_2, 0.75)

Example 1 Assume $m = 2$, $k = 1$, $f(x) = 0.5 \times s_1(x) + 0.5 \times s_2(x)$. The dataset is illustrated in Table 1, where i is from 3 to $n - 2$. Here we set $b = 10$ in DTA.

Obviously the top-1 object is R_1 and TA performs only 4 sorted accesses and 4 random accesses. However, since the lean value of the available section in L_1 is always smaller than that in L_2, L_2 is chosen to be accessed all the time. Therefore, DTA calls $\left\lceil \frac{n-4}{2} \right\rceil + 3$ sorted accesses and as many random accesses. Since n could be arbitrary large, algorithm DTA is not an instance optimal algorithm. □

The reason DTA is not instance optimal is that DTA "selects" some lists to access instead of all lists in parallel like TA. By doing this selection, DTA may do fewer accesses in most databases but may "miss" some important information in some particular databases like Example 1. Nevertheless, we can force DTA to be instance optimal by a little modify of DTA. We call the modified algorithm "Selective- Density Threshold Algorithm (i.e. S-DTA)", which is described by the pseudo-code in Fig. 3.

We note the access cost of S-DTA is at most v times as TA where v is a positive integer. So S-DTA is instance optimal. Since the optimal ratio of TA is $m + m(m-1)c_R / c_S$, the optimal ratio of S-DTA is $v(m + m(m-1)c_R / c_S)$ under some natural assumption [3].

V. EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our algorithms. Our algorithms are implemented in C/C++ language. We perform our experiments on a 2-CPU server with 8GB shared memory and each CPU is 4-core Intel Xeon E5430 2.66GHz.

We do experiments on three synthetic datasets and all generated local scores belong to the interval [0, 1]. The three synthetic datasets are produced to model different input scenarios; they are UI, NI and CO, respectively. All of them are generated using the same methodology in [5] [7] [10].

For synthetic datasets, our default settings for different parameters are shown in Table 2. We choose m to be the varying parameter in one of our tests. As a result, the aggregate function $f(x)$ ought to vary with m. Since DTA and S-DTA are designed to determine that which attribute list should be accessed in the query, we should distinguish the attributes with different aggregate weights. Typically, we set

Selective -Density Threshold Algorithm (S-DTA)

Pre-computing Phase:

Build the Density Index of the given database.

Computing Phase:

1: $Y = \emptyset$, $\tau = 0$, $M_k = 0$, $select = 0$, $\underline{s_i} = 1$ where $i = 1, 2 ...m$.
2: **while** ($Y.size < k$ or $\tau > M_k$) **do**
3: … // the same as line 3 in DTA.
4: **for** item $(x, s_i(x))$ = **from** the head item **down to** the tail item of Sn_j **do**
5: $select = select + 1$.
6: **if** $select$ mod v == 0 **then**
7: **for** each $l \in \{1, 2 \dots m\}$ **do in parallel**
8: $(x, s_l(x))$ = Get_next_item $(L_1, \dots, L_m)$.
// get next item from one of the lists $L_1, \dots, L_m$ in parallel.
9: $\underline{s_l} = s_l(x)$.
10: … // the same as DTA from line 6 to line 17.
11: **end for.**
12: **else**
13: … // the same as DTA from line 5 to line 17.
14: **end for.**
15: **end while.**
16: **Return** Y.

Figure 3. Selective-Density Threshold Algorithm

the aggregate function with Fibonacci numbers Fi = 1, 1, 2, 3, 5 … :

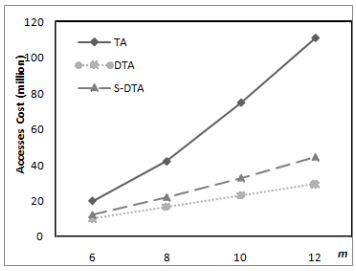
$$f(x) = \sum_{i=1}^{m} \frac{F_i}{SUM} \bullet s_i(x), \text{ where } SUM = \sum_{i=1}^{m} F_i .$$

As c_R is usually much higher than c_S, we assume c_R =10 and c_S =1 in our tests. Therefore, the access cost of the queries is $a_S + 10a_R$.

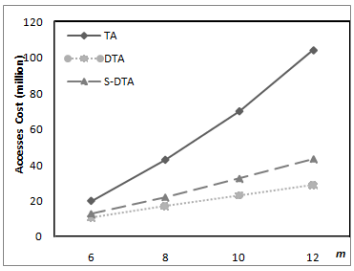
Now we show the experimental results in the figures below:

Fig. 4 illustrates the experimental results in which the parameters are set as those in Table 2 except that m, which varies from 6 to 12 with step 2. The results over the databases we considered show us that our algorithms perform more efficient compared with TA as the m becoming larger. This is mainly because the selection on accessing the lists in DTA and S-DTA makes great difference on the efficiency over datasets with many attributes while parallel accessing become weaker as a result of its neglect of the distinction between the attributes. Moreover, we note that the access cost of S-DTA is always higher than DTA and lower than TA. However, over some databases such as Example 1, S-DTA performs much lower than DTA while TA turns out to be the most efficient algorithm in this case. Obviously, since S-DTA selectively calls DTA (mainly) and TA in the query, it combines the efficiency of DTA and the instance optimality of TA. Therefore, the access cost of S-DTA must be between those of TA and DTA.

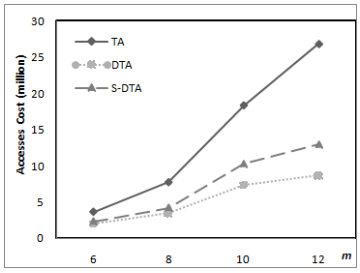
The experimental results shown in Fig. 5 where the parameter k varies while the others are set as those in Table 2 indicate the access costs of our algorithms are significantly lower than that of TA over the considered datasets. Besides, the access costs increase much more slightly than that when m is the changed parameter. Furthermore, we can also observe



(a) over UI

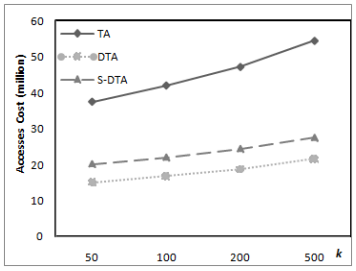


(b) over NI

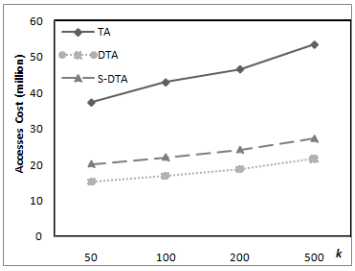


(c) over CO

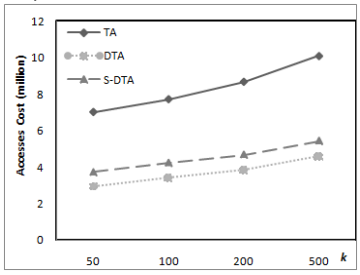
Figure 4. Access Cost v.s. *m* over synthetic datasets, *n*=1,000,000, *k*=100, *b*=1000, *v*=10



(a) over UI



(b) over NI



(c) over CO

Figure 5. Access Cost v.s. *k* over synthetic datasets, *n*=1,000,000, *m*=8, *b*=1000, *v*=10

that the access cost of S-DTA keeps between those of TA and DTA.

VI. Conclusions

Based on the observation on the TA algorithm, we proposed the novel and efficient Density Threshold Algorithm (DTA) to minimize the useless accesses of a top-k query and introduced an efficient indexing structure called Density Index to support our algorithms. Thereafter, we proved the DTA is not instance optimal in Fagin's notion and we also turned the DTA into an instance optimal algorithm named Selective-Density Threshold Algorithm (S-DTA). Finally, we did extensive experiments to compare our algorithms and the TA algorithm. The results showed that our algorithms had significant improvement on the performance of TA algorithm.

Acknowledgment

This work is supported by the National Science Foundation of China under the grant [No. 60873210].

References

[1] I. Ilyas, G. Beskales, M. A. Soliman. "A Survey of Top-k Query Processing Techniques in Relational Database Systems," *ACM Computing Surveys*, 2008.

[2] J. Yuan, G. Z. Sun, Y. Tian, G. Chen and Z. Liu. "Selective-NRA Algorithms for Top-k Queries," *APWeb/ WAIM* 2009.

[3] R. Fagin, A. Lotem M. Naor. "Optimal aggregation algorithms for middleware," *PODS*, 2001.

[4] U. Güntzer, W. T. Balke, W. Kießling. "Optimizing Multi-Feature Queries for Image Databases," *26TH VLDB*, 2000.

[5] Z. Q. Gong, G. Z. Sun, J. Yuan,Y. Zhong. "Efficient Top-*k* Query Algorithms Using *K*-skyband Partition," *INFOSCALE*, 2009.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[7] Z. Q. Gong, G. Z. Sun, D. Q. Chen. "Parallel Algorithms for Top-k Query Processing," Unpublished .

[8] I. Ilyas, R. Shah, W. Aref, J. Vitter, A. Elmagarmid. "Rank-Aware Query Optimization," *ACM SIGMOD*, 2004.

[9] R. Fagin. "Combining fuzzy information from multiple systems," *J. Comput. System Sci*, 58 (1), 1999.

[10] D. Q. Chen, G. Z. Sun, Z. Q. Gong, J. Yuan. "Efficient Approximate Top-k Query Algorithm Using Cube Index," Unpublished.

[11] R. Fagin, R. Kumar, D. Sivakumar. "Comparing Top K Lists," *ACM-SIAM SODA*, 2003.

[12] S. Chaudhuri, N. Dalvi and R. Kaushik. "Robust Cardinality and Cost Estimation for Skyline Operator," *ICDE*, 2006.