

# Efficient Top- $k$ Query Algorithms Using $K$ -skyband Partition

Zhenqiang Gong, Guang-Zhong Sun\*, Jing Yuan, and Yanjing Zhong

MOE-MS Key Laboratory of Multimedia Computing and Communication,  
School of Computer Science and Technology,  
University of Science and Technology of China, Hefei, 230027, P.R. China

{gzqiang, yuanjing, zhongyj}@mail.ustc.edu.cn; gzsun@ustc.edu.cn

**Abstract.** Efficient processing of top- $k$  queries has become a classical research area. Fagin et al. proposed the “middleware cost” for a top- $k$  query algorithm. In some scenario, there is no way to perform a random access, and Fagin et al. proposed NRA (No Random Access) algorithm for that. In this paper, we investigate the intrinsic relation between top- $k$  queries and  $K$ -skyband queries. Based on that relation, we propose a novel algorithm DNRA (Dominate-NRA). The main idea of DNRA is to partition the original dataset into two sub-datasets depending on whether they belong to  $K$ -skyband or not. We prove that DNRA performs no more sorted accesses than NRA on any dataset. Furthermore, we partition the dataset into  $N$  sub-datasets ( $N$  is the number of objects in the dataset), and then we propose our algorithm ADNRA (Advanced-DNRA). The partition of the dataset is pre-computed, and we discuss two techniques to fulfill it. Extensive experiments show that our algorithms perform several orders of magnitude fewer accesses than NRA and that ADNRA performs significantly fewer accesses than DNRA on some datasets.

**Keywords** Top- $k$  Queries,  $K$ -skyband Queries, NRA Algorithm, Dominate

## 1 Introduction

Assume there are a huge amount of objects and every object has  $M$  attributes, for each attribute the object has a local score. These local scores can be aggregated to a total score by an aggregate function, and we want to know which  $k$  objects have the largest total scores. This scenario is generalized as “top- $k$  queries”.

Top- $k$  queries have attracted considerable attention because of its wide use in many areas such as information retrieval[6][7], network and system monitoring[8][9], P2P systems and sensor networks [10][11], etc. 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.[1] is that the dataset consists of  $M$  sorted lists with  $N$  data items. Each data item can be accessed through *sorted access* or *random access*. However, top- $k$  queries with no random access have

---

\* Corresponding author

received increasing interests because in many applications random access is impossible or it's much more expensive than sorted access [2][3][4]. Fagin et al. proposed NRA (No Random Access) algorithm for this case. In this paper, we focus on the cases with no random access, but our methods can be easily generalized to cases where random access is possible.

$K$ -skyband[14] computation is another kind of query, which returns those objects that are dominated by at most  $K-1$  other objects. An object dominates another object if it is as good or better in all attributes and better in at least one attribute. The essential difference between top- $k$  queries and  $K$ -skyband queries is that top- $k$  answers may vary with the different aggregate functions while  $K$ -skyband answers do not.

Actually, there exist intrinsic connections between top- $k$  queries and  $K$ -skyband queries. Firstly, in this paper, we investigate the relation between top- $k$  queries and  $K$ -skyband queries, i.e. top- $k$  answers for any increasingly monotone aggregate function belong to  $K$ -skyband, where  $k \leq K$ . Secondly, based on the investigation, we propose our algorithm DNRA (Domination-NRA). The main idea of DNRA is to partition the original dataset into two sub-datasets depending on whether they belong to the  $K$ -skyband or not and it only accesses the  $K$ -skyband objects when answering top- $k$  queries. For any dataset instance, we prove that DNRA performs no more sorted accesses than NRA. Thirdly, motivated by the idea of partitioning the dataset, we take a further step to partition the dataset into  $N$  sub-datasets (some sub-datasets may be empty) according to the *degree of domination* (see definition 3 in section 3.3) of the objects. Our algorithm ADNRA (Advanced-DNRA) comes into being on the basis of this partition. Fourthly, we discuss two techniques of dataset partition, which is done *offline*. Finally, we do extensive experiments to compare NRA algorithm and our algorithms. The results show that our algorithms perform several orders of magnitude fewer sorted accesses than NRA and that ADNRA performs significantly fewer sorted accesses than DNRA.

The rest of this paper is organized as follows. In section 2, we define the problem formally and review NRA algorithm. In section 3, we describe our algorithm DNRA and its advanced version ADNRA, and we discuss the pre-computation of the dataset partition. In section 4, we show the experimental results. In section 5, we discuss some related works. Finally, in section 6, we conclude this paper and introduce our future works.

## 2 Problem Definitions and NRA Algorithm

In this section, we describe the model of our problem and review NRA algorithm proposed in [1].

Our model of the dataset can be described as follows: assume the dataset  $D$  consists of  $M$  sorted lists, which are denoted as  $L_1, L_2, \dots, L_M$ . Each sorted list consists of  $N$  data items. Each data item is a pair  $(x, s_i(x))$ , where  $x$  is an object,  $s_i(x)$  is  $x$ 's  $i$ th local score which is a real number in the interval  $[0, 1]$ . Sorted list means that objects in each list are sorted in descending orders according to their local scores. Each data item can be accessed only by sorted access, so the middleware cost of an algorithm is  $a_s C_s$ , where  $a_s$  is the number of sorted accesses performed and  $C_s$  is the cost of a

single sorted access. For any object  $x$ , its overall score  $S_x = f(s_1(x), s_2(x), \dots, s_M(x))$ , where the aggregate function  $f$  is assumed to be monotone in this paper. Our task is to find  $k$  objects whose overall scores are the highest  $k$  ones.

**Table 1.** Meanings of basic symbols used.

$N$	Number of objects	$k$	Number of objects returned
$M$	Number of lists	$S_x^{ub}$	upper bound of $S_x$
$S_x$	overall score of $x$	$s_i(x)$	The $i$ th local score of $x$
$S_x^{lb}$	lower bound of $S_x$	$s_i$	Bottom score of the $i$ th list

For this problem, Fagin et al. proposed the algorithm NRA (No Random Access)[1]. The basic idea of NRA is to evaluate an object's overall score using the upper bound and lower bound of the overall score. We rewrite NRA algorithm in Fig.1. In the algorithm, we replace the original notations with ours.

---

**Algorithm NRA:**

1. Do sorted access in parallel to each of the  $M$  sorted lists  $L_i$ . At each depth  $d$  (when  $d$  objects have been accessed under sorted access in each list):
    - Maintain the bottom values  $s_i^{(d)}$ ,  $i \in \{1, 2, \dots, M\}$ , encountered in the lists.
    - For every object  $x$  with discovered fields  $L = L^{(d)}(x) \subseteq \{1, \dots, M\}$ , compute the values  $S_x^{lb}$  and  $S_x^{ub}$ . (For object  $x$  that has not been seen, these values are virtually computed as  $S_x^{lb} = f(0, 0, \dots, 0)$  and  $S_x^{ub} = f(s_1^{(d)}, s_2^{(d)}, \dots, s_M^{(d)})$  which is the threshold value.)
    - Let  $Y^{(d)}$ , the current top- $k$  list, contain the  $k$  objects with the largest  $S_x^{lb}$  values seen so far (and their scores); if two objects have the same  $S_x^{lb}$  value, then ties are broken using the  $S_x^{ub}$  values, such that object with the highest  $S_x^{ub}$  value wins (and arbitrarily among objects that tie for the highest  $S_x^{ub}$  value). Let  $t^{(d)} = \min \{ S_x^{lb} \mid x \in Y^{(d)} \}$
  2. Halt when (a) at least  $k$  distinct objects have been seen and (b)  $\tilde{t} \leq t^{(d)}$ , where  $\tilde{t} = \max \{ S_x^{ub} \mid x \notin Y^{(d)} \}$ . Return the objects in  $Y^{(d)}$ .
- 

**Fig. 1.** Algorithm NRA

The following example illustrates NRA algorithm.

**Example 1.** Assume  $M=2$ ,  $N=6$ ,  $k=2$ , the aggregation function is summation, and the lists shown in Table 2 of Fig.2 can only be sorted accessed. Call accessing each list once in parallel an *attempt*. In Fig.2, Table 3-6 show how algorithm NRA performs sorted accesses at each attempt on this dataset. Table 7 shows how the top-2 objects and the parameter  $\tilde{t}$  updated at each attempt. NRA algorithm halts after 8 sorted accesses

**Table 2.** Sorted Lists

(X <sub>2</sub> ,0.95)	(X <sub>3</sub> ,0.95)
(X <sub>1</sub> ,0.92)	(X <sub>4</sub> ,0.90)
(X <sub>5</sub> ,0.89)	(X <sub>6</sub> ,0.88)
(X <sub>3</sub> ,0.88)	(X <sub>2</sub> ,0.87)
(X <sub>4</sub> ,0.87)	(X <sub>1</sub> ,0.87)
(X <sub>6</sub> ,0.86)	(X <sub>5</sub> ,0.85)

**Table 3.** Attempt 1

<b>(X<sub>2</sub>,0.95)</b>	<b>(X<sub>3</sub>,0.95)</b>
(X <sub>1</sub> ,0.92)	(X <sub>4</sub> ,0.90)
(X <sub>5</sub> ,0.89)	(X <sub>6</sub> ,0.88)
(X <sub>3</sub> ,0.88)	(X <sub>2</sub> ,0.87)
(X <sub>4</sub> ,0.87)	(X <sub>1</sub> ,0.87)
(X <sub>6</sub> ,0.86)	(X <sub>5</sub> ,0.85)

**Table 4.** Attempt 2

<b>(X<sub>2</sub>,0.95)</b>	<b>(X<sub>3</sub>,0.95)</b>
<b>(X<sub>1</sub>,0.92)</b>	<b>(X<sub>4</sub>,0.90)</b>
(X <sub>5</sub> ,0.89)	(X <sub>6</sub> ,0.88)
(X <sub>3</sub> ,0.88)	(X <sub>2</sub> ,0.87)
(X <sub>4</sub> ,0.87)	(X <sub>1</sub> ,0.87)
(X <sub>6</sub> ,0.86)	(X <sub>5</sub> ,0.85)

**Table 5.** Attempt 3

<b>(X<sub>2</sub>,0.95)</b>	<b>(X<sub>3</sub>,0.95)</b>
<b>(X<sub>1</sub>,0.92)</b>	<b>(X<sub>4</sub>,0.90)</b>
<b>(X<sub>5</sub>,0.89)</b>	<b>(X<sub>6</sub>,0.88)</b>
(X <sub>3</sub> ,0.88)	(X <sub>2</sub> ,0.87)
(X <sub>4</sub> ,0.87)	(X <sub>1</sub> ,0.87)
(X <sub>6</sub> ,0.86)	(X <sub>5</sub> ,0.85)

**Table 6.** Attempt 4

<b>(X<sub>2</sub>,0.95)</b>	<b>(X<sub>3</sub>,0.95)</b>
<b>(X<sub>1</sub>,0.92)</b>	<b>(X<sub>4</sub>,0.90)</b>
<b>(X<sub>5</sub>,0.89)</b>	<b>(X<sub>6</sub>,0.88)</b>
<b>(X<sub>3</sub>,0.88)</b>	<b>(X<sub>2</sub>,0.87)</b>
(X <sub>4</sub> ,0.87)	(X <sub>1</sub> ,0.87)
(X <sub>6</sub> ,0.86)	(X <sub>5</sub> ,0.85)

**Table 7.** Each attempt of NRA

Top-2		$\tilde{t}$
Objects	Lower bounds	
(X <sub>2</sub> , X <sub>3</sub> )	(0.95,0.95)	1.90
(X <sub>2</sub> ,X <sub>3</sub> )	(0.95,0.95)	1.83
(X <sub>2</sub> ,X <sub>3</sub> )	(0.95,0.95)	1.80
(X <sub>2</sub> ,X <sub>3</sub> )	(1.82,1.83)	1.79

**Fig. 2.** An example shows how NRA works

### 3 Dominate-NRA Algorithms

In this section, we first discuss the relation between top- $k$  queries and  $K$ -skyband queries (Section 3.1). Thereafter, in Section 3.2, we describe our algorithm DNRA and discuss its performance. Then, in Section 3.3, we introduce ADNRA. Finally, in Section 3.4, we address the process of pre-computation.

#### 3.1 Top- $k$ Queries and $K$ -skyband Queries

In the following, we first introduce the definition of dominate and propose an observation about it. Then, we define the  $K$ -skyband set and discuss its relation to top- $k$  queries.

**Definition 1** *dominate*[5]. We say object  $x$  *dominates*  $y$  or  $y$  is *dominated* by  $x$  if and only if they satisfy two conditions: (1) for each  $i \in \{1, 2, \dots, M\}$ ,  $s_i(x) \geq s_i(y)$ . (2) there exists at least one number  $j \in \{1, 2, \dots, M\}$  satisfying  $s_j(x) > s_j(y)$ .

Our definition of dominate is different from that in [5], since we use  $\geq$  (or  $>$ ) instead of  $\leq$  (or  $<$ ). However, there does not exist essential differences between them

**Observation 1:** If object  $x$  dominates object  $y$  and the aggregate function is increasingly monotone, then we have  $s_x > s_y$ .

**Proof:** We can easily get the correctness of the observation according to the definition of dominate.  $\square$

**Definition 2** *K-skyband* [14]. For a dataset  $D$ , its *K-skyband* is a set of objects that are dominated by at most  $K-1$  other objects.

For the dataset in Example 1, the *K-skyband* for  $K=2$  includes objects  $x_1, x_2, x_3$  and  $x_4$ . Notice that  $x_2$  and  $x_3$ , which are the top-2 objects with  $f = \text{sum}$ , belongs to the *K-skyband* for  $K=2$ .

**Observation 2:** For any increasingly monotone aggregate function, the top- $k$  objects belong to the *K-skyband*, where  $k \leq K$ .

**Proof:** For any object  $x$  that does not belong to the *K-skyband*, there exists at least  $K$  other objects that dominate it. According to observation 1, we know that there exists at least  $K$  other objects whose overall scores are strictly larger than  $x$ 's, which means that  $x$  cannot be top- $K$ . Since  $k \leq K$ , we have the top- $k$  objects for any increasingly monotone aggregate function belong to the *K-skyband*.  $\square$

Motivated by the fact that the top- $k$  objects for any increasingly monotone aggregate function belong to the *K-skyband*, where  $k \leq K$ , we can use the *K-skyband* query as a pre-computing step to answer top- $k$  queries. After pre-computing the *K-skyband* offline, any top- $k$  queries with  $k \leq K$  for any increasingly monotone aggregate function can be addressed only accessing the *K-skyband* objects.

### 3.2 Dominate-NRA Algorithm(DNRA)

In this section, based on the observations discussed in the above section, we first introduce our algorithm DNRA (Dominate-NRA) and prove its correctness. Then, we evaluate its performance. Specifically, we prove that DNRA performs no more sorted accesses than NRA on any dataset.

The description of DNRA algorithm is shown in Fig.3. Through pre-computation (addressed in section 3.4), it first partitions the original dataset  $D$  into two sub-datasets, i.e.  $D_g$  and  $D_b$ , where  $D_g$  contains all the *K-skyband* objects and  $D_b$  contains the

rest. Then it runs NRA only on  $D_g$  in order to find top- $k$  answerers of the original dataset  $D$ .

---

**Algorithm DNRA:**

1. Through pre-computation, we partition the original dataset  $D$  into two sub-datasets, i.e.  $D_g$  and  $D_b$ , where  $D_g$  contains all the  $K$ -skyband objects and  $D_b$  contains the rest.
  2. Run NRA algorithm on  $D_g$  to find top- $k$  objects, where  $k \leq K$ .
- 

**Fig. 3.** Algorithm DNRA

The following theorem provides the correctness of DNRA.

**Theorem 1** *If the aggregate function is increasingly monotone, then DNRA correctly finds the top- $k$  objects, where  $k \leq K$ .*

**Proof:** According to observation 2, top- $k$  objects belong to the  $K$ -skyband, where  $k \leq K$ . Here,  $D_g$  and the  $K$ -skyband contain the same objects, that is, top- $k$  objects belong to  $D_g$ . Since NRA algorithm correctly finds the top- $k$  objects in  $D_g$ , DNRA correctly finds the top- $k$  objects of the original dataset.  $\square$

To evaluate the performance of DNRA, we first introduce three lemmas as follows.

**Lemma 1** Let  $DS_x^{ub}(d)$  be the upper bound of  $x$ 's overall score in DNRA at depth  $d$ . Let  $NS_x^{ub}(d)$  be the upper bound of  $x$ 's overall score in NRA at depth  $d$ . Then, for any object  $x \in D_g$ , we have  $DS_x^{ub}(d) \leq NS_x^{ub}(d)$ .

**Proof:** The upper bound of object  $x$ 's overall score is calculated using an increasingly monotone aggregate function by substituting its missing fields with relative bottom values. For any object  $x \in D_g$ , its fields seen by NRA before depth  $d$  are also seen by DNRA before depth  $d$ , and its fields seen by DNRA but not seen by NRA before depth  $d$  are no larger than the relative bottom values in NRA. Furthermore, bottom values in DNRA are no larger than the relative bottom values in NRA at depth  $d$ . So, we have  $DS_x^{ub}(d) \leq NS_x^{ub}(d)$ .  $\square$

**Lemma 2** Let  $DS_x^{lb}(d)$  be the lower bound of  $x$ 's overall score in DNRA at depth  $d$ . Let  $NS_x^{lb}(d)$  be the lower bound of  $x$ 's overall score in NRA at depth  $d$ . Then, for any object  $x \in D_g$ , we have  $DS_x^{lb}(d) \geq NS_x^{lb}(d)$ .

**Proof:** For any object  $x \in D_g$ , its fields seen by NRA before depth  $d$  are also seen by DNRA before depth  $d$ . The lower bound of  $x$ 's overall score is calculated using an increasingly monotone aggregate function by substituting its missing fields with 0. So we have  $DS_x^{lb}(d) \geq NS_x^{lb}(d)$ .  $\square$

**Lemma 3** Let  $Y(d)$  contain the current top- $k$  objects in NRA at depth  $d$  and  $t(d) = \min \{ NS_x^{lb}(d) \mid x \in Y(d) \}$ . Let  $Y'(d)$  contain the current top- $k$  objects in DNRA and  $i'(d) = \min \{ DS_x^{lb}(d) \mid x \in Y'(d) \}$ . If all the objects in  $Y(d)$  belong to  $D_g$ , then we have  $i'(d) \geq t(d)$ .

**Proof:** At any depth  $d$ , assume  $x' \in Y'(d)$  satisfying  $DS_{x'}^{lb}(d) = i'(d)$ . There are two cases depending on whether  $x'$  belongs to  $Y(d)$ .

Case 1:  $x' \in Y(d)$ . Since  $x' \in Y(d)$ , we have  $NS_{x'}^{lb}(d) \geq t(d)$ . And according to lemma 2, we have  $DS_{x'}^{lb}(d) \geq NS_{x'}^{lb}(d)$ . So, we have  $i'(d) = DS_{x'}^{lb}(d) \geq NS_{x'}^{lb}(d) \geq t(d)$ , that is,  $i'(d) \geq t(d)$ , as desired.

Case 2:  $x' \notin Y(d)$ . In this case, there exists one object  $y \in Y(d)$  but  $y \notin Y'(d)$ . Since  $y \in Y(d)$ , we have  $NS_y^{lb}(d) \geq t(d)$ . Since  $y \notin Y'(d)$ , we have  $i'(d) \geq DS_y^{lb}(d)$ . And according to lemma 2, we have  $DS_y^{lb}(d) \geq NS_y^{lb}(d)$ . So, we get  $i'(d) \geq DS_y^{lb}(d) \geq NS_y^{lb}(d) \geq t(d)$ , that is,  $i'(d) \geq t(d)$ , as desired.  $\square$

**Theorem 2** The number of sorted accesses to the lists done by DNRA is always less than or equal to that of NRA.

**Proof:** For any dataset  $D$ , we assume NRA algorithm stops at depth  $d$  and DNRA algorithm does not stop before depth  $d$ . Let  $Y(d)$  contain the top- $k$  objects that outputted by NRA and  $t(d) = \min \{ NS_x^{lb}(d) \mid x \in Y(d) \}$ . Let  $Y'(d)$  contain the current top- $k$  objects in DNRA at depth  $d$  and  $i'(d) = \min \{ DS_x^{lb}(d) \mid x \in Y'(d) \}$ . The  $k$  objects in  $Y(d)$  must belong to  $D_g$ , so we have, according to lemma 3,  $i'(d) \geq t(d)$ . Since we assume DNRA does not stop at depth  $d$ , there exists an object  $x \in D_g$  and  $x \notin Y'(d)$  satisfying that  $DS_x^{ub}(d) > i'(d)$ . We need to generate contradictions. There are two cases depending on whether  $x$  belongs to  $Y(d)$ .

Case 1:  $x \notin Y(d)$ . Since we have  $DS_x^{ub}(d) \leq NS_x^{ub}(d)$  according to lemma 1 and  $NS_x^{ub}(d) \leq t(d)$  according to the stopping rule of NRA, we get  $DS_x^{ub}(d) \leq NS_x^{ub}(d) \leq t(d) \leq i'(d)$ , which contradicts with  $DS_x^{ub}(d) > i'(d)$ .

Case 2:  $x \in Y(d)$ . In this case, there exists one object  $y \in Y'(d)$  but  $y \notin Y(d)$ . Since  $y \in Y'(d)$  and  $x \in Y(d)$ , we have  $DS_y^{lb}(d) \geq i'(d) \geq DS_x^{lb}(d)$ . Since  $x \in Y(d)$  and  $y \notin Y(d)$ , we have  $NS_x^{lb}(d) \geq t(d) \geq NS_y^{ub}(d)$ . Combined with lemma 1 and lemma 2, we get  $NS_x^{lb}(d) \geq t(d) \geq NS_y^{ub}(d) \geq DS_y^{ub}(d) \geq DS_y^{lb}(d) \geq i'(d) \geq DS_x^{lb}(d) \geq NS_x^{lb}(d)$ . So we have  $DS_x^{lb}(d) = DS_y^{lb}(d) =$

$DS_y^{ub}(d) = t'(d)$ . Since  $DS_x^{lb}(d) = DS_y^{lb}(d)$ , we have  $DS_x^{ub}(d) \leq DS_y^{ub}(d)$  according to the ties breaking rules of  $Y(d)$ . Finally, we get  $t'(d) = DS_x^{lb}(d) \leq DS_x^{ub}(d) \leq DS_y^{ub}(d) = t'(d)$ , that is,  $DS_x^{ub}(d) = t'(d)$ , which contradicts with  $DS_x^{ub}(d) > t'(d)$ .

Hence, the assumption that DNRA does not stop before depth  $d$  is unwarranted. In other words, DNRA does stop before depth  $d$ .  $\square$

### 3.3 Optimization

The key idea of DNRA is that it classifies the objects into two categories depending on whether they belong to the  $K$ -skyband or not and it only accesses the  $K$ -skyband objects when answering top- $k$  queries. One weakness of DNRA is that it cannot address the top- $k$  queries with  $k > K$ . To cope with top- $k$  queries with  $k > K$ , we take a further step to classify the objects into  $N$  categories. In the following, we begin by proposing a method to classify objects. Then, based on this method, we partition the original dataset  $N$  sub-datasets. Our algorithm ADNRA (Advanced-DNRA) comes into being on the basis of this partition.

**Definition 3** *degree of domination.* If some object  $x$  is dominated by  $i$  other objects, we say the degree of domination of  $x$  is  $i$ , denoted as  $dd(x) = i$ .

This definition provides us a kind of method to classify objects, i.e. we can classify objects into  $N$  categories by their degree of domination. Based on the classification of the objects, we can partition the original dataset  $D$  into  $N$  sub-datasets, denoted as  $D_0, D_1, \dots, D_{N-1}$  (some sub-datasets may be empty), where  $\forall x \in D_i$ , satisfying  $dd(x) = i$ . The partition is done offline. The following theorem reduces our accessing scope to  $D_0, D_1, \dots, D_{k-1}$  when answering top- $k$  queries.

**Theorem 3** *If the aggregate function is increasingly monotone, then top- $k$  objects must be among sub-datasets  $D_0, D_1, \dots, D_{k-1}$*

**Proof:** Actually, the  $K$ -skyband for  $K=k$  consists of sub-datasets  $D_0, D_1, \dots, D_{k-1}$ . And according to observation 2, top- $k$  objects belong to  $K$ -skyband if  $k \leq K$ . So the top- $k$  objects must be among sub-datasets  $D_0, D_1, \dots, D_{k-1}$ .  $\square$

Now, we can describe our algorithm ADNRA. Its description is shown in Fig.4. ADNRA algorithm uses a sequential policy to access  $D_0, D_1, \dots, D_{k-1}$ . Specifically, when  $candidate_j = \emptyset$  and  $t \geq T_j$ , ADNRA stops accessing  $D_j$  and goes on accessing  $D_{j+1}$  (Note:  $candidate_j$  may return to be nonempty since some object in  $Y$  coming from  $D_j$  may become a candidate some time later). After this sequential process, it checks whether there exists some  $candidate$  that is nonempty. And if  $candidate_j$  is nonempty, it continues performing sorted accesses on  $D_j$ .

Actually, we can design another policy to perform sorted accesses on sub-datasets  $D_0, D_1, \dots, D_{k-1}$ , i.e. dataset-level-parallel policy. Dataset-level-parallel policy works as

follows: firstly, we distribute the sub-datasets  $D_0, D_1, \dots, D_{k-1}$  onto  $k$  processors; secondly, let the  $i$ th processor compute in parallel the top- $(k-i)$  objects in  $D_i$ , and store them in  $Y_i$ ; thirdly, we combine  $Y_0, Y_1, \dots, Y_{k-1}$  to get  $Y$ ; finally, it checks whether there exists some *candidate* that is nonempty. And if  $candidate_i$  is nonempty, it continues performing sorted accesses on  $D_i$  in parallel.

---

**Algorithm ADNRA:**

1. Run NRA on  $D_0$  to find top- $k$  objects in  $D_0$ , and store the results in  $Y$ . Let  $t = \min\{S_x^{lb} \mid x \in Y\}$
  2.  $j := 1$
  3. For  $D_j$ , do sorted accesses in parallel to each of the  $M$  sorted lists  $L_i$ . Accessing each list once on  $D_j$  in parallel is called an *attempt*. After each attempt on  $D_j$ , go to *update\_process*: update  $s_{1j}, s_{2j}, \dots, s_{Mj}$  (bottom scores of  $D_j$ ),  $T_j$  (the threshold value of  $D_j$ ), lower bounds and upper bounds of objects in  $candidate_j$  ( $candidate_j$  is a set of objects whose current upper bounds of their scores are larger than  $t$  and that do not belong to  $Y$ ) and of objects in  $Y$  that come from  $D_j, Y, t$  and all the sets of  $candidate$ . When  $candidate_j = \emptyset$  and  $t \geq T_j$ , go to step 4.
  4.  $j := j + 1$ . If  $j < k$ , then go to step 3, or go to step 5.
  5. For each  $l$ , while  $candidate_l \neq \emptyset$ , continue performing sorted accesses on  $D_l$  in parallel. After each attempt on  $D_l$ , go to *update\_process* as described in step 3.
  6. Halt when (a) at least  $k$  distinct objects have been seen, (b)  $t \geq T_j$ , for  $j = 0, 1, \dots, k-1$  and (c) for  $j = 0, 1, \dots, k-1$ ,  $candidate_j = \emptyset$ . Return the objects in  $Y$ .
- 

**Fig. 4.** Algorithm ADNRA

The following theorem provides the correctness of ADNRA.

**Theorem 4** *If the aggregate function is increasingly monotone, then ADNRA correctly finds the top- $k$  objects.*

**Proof:** Let  $t = \min\{S_x^{lb} \mid x \in Y\}$ . We must show that for any object  $x$ ,  $x \notin Y$  and  $x \in D_j$  for some  $j \in \{0, 1, \dots, k-1\}$ , we have  $S_x \leq t$ . There are two cases, depending on whether  $x$  has been seen or not when ADNRA stops.

Case 1:  $x$  has been seen when ADNRA stops. Since  $candidate_j = \emptyset$  when ADNRA stops, we have  $S_x \leq S_x^{ub} \leq t$ , as desired.

Case 2:  $x$  hasn't been seen when ADNRA stops. Since  $S_x^{ub} = T_j$  and  $t \geq T_j$  when ADNRA stops, we have  $S_x \leq S_x^{ub} = T_j \leq t$ , as desired.  $\square$

The following example illustrates how ADNRA works.

**Example 2.** The dataset is the same as that in Example 1. Through pre-computation (addressed in section 3.4), the dataset is partitioned into three sub-datasets, which are illustrated in Table 8. First, ADNRA runs NRA on  $D_0$  to find top-2 objects in  $D_0$ , that is,  $x_2$  and  $x_3$ . Then it continues accessing  $D_1$ . After just an attempt on  $D_1$ , it stops accessing  $D_1$ . Then it checks whether there exists some number  $l$  satisfying  $candidate_l \neq \emptyset$ , if there does, it continues accessing  $D_l$ . In this example, there does not exist such number  $l$ . So ADNRA halts after 6 sorted accesses. ADNRA performs two fewer sorted accesses than NRA on this dataset.

**Table 8.** Sub-datasets

$D_0$	
$(X_2, 0.95)$	$(X_3, 0.95)$
$(X_3, 0.88)$	$(X_2, 0.87)$
$D_1$	
$(X_1, 0.92)$	$(X_4, 0.90)$
$(X_4, 0.87)$	$(X_1, 0.87)$
$D_2$	
$(X_5, 0.89)$	$(X_6, 0.88)$
$(X_6, 0.86)$	$(X_5, 0.85)$

### 3.4 Pre-computation

Both our algorithms are based on the pre-computation, which is aimed to pre-compute  $D_b$  and  $D_g$  in DNRA and  $D_0, D_1, \dots, D_{N-1}$  in ADNRA.  $D_g$  contains and only contains the  $K$ -skyband objects, so any algorithm that can address  $K$ -skyband queries, e.g. BBS [14], can be used to pre-compute  $D_b$  and  $D_g$  in DNRA. For the pre-computation of  $D_0, D_1, \dots, D_{N-1}$  in ADNRA, the key point is to pre-compute the degree of domination of each object. In the rest of this section, we propose an algorithm called BFA (Brute-force algorithm) and introduce an algorithm called bitmap to pre-compute the degree of domination of each object. Bitmap algorithm is proposed in [13] to address skyline queries, but it is also powerful to compute the degree of domination of each object.

#### 3.4.1 Brute-force Algorithm

Brute-force algorithm (BFA) scans the  $M$  sorted lists, and store the objects with their  $M$  local scores in an extra list. The data item in the extra list is like  $(x, s_1(x), s_2(x), \dots, s_M(x))$ , where  $x$  is an object,  $s_i(x)$  is  $x$ 's  $i$ th local score. The objects in the extra list are sorted in non-increasing orders respect to their first local scores. Then BFA scans the sorted extra list. When an object  $x$  is seen, BFA compares it with objects seen before it and objects having not been seen yet but whose first

local score is the same as  $x$ 's. After the comparisons, we get the degree of domination of  $x$ . The time cost of BFA is  $O(MN^2)$ . And the extra space needed is also  $O(MN^2)$ .

### 3.4.2 Bitmap

This technique encodes in bitmaps all the information required to decide the degree of domination of all the objects. An object  $x$  whose local scores are  $s_1(x), s_2(x), \dots, s_M(x)$  is mapped to an  $n$ -bit vector, where  $n$  is the total number of distinct values over all lists. Let  $n_i$  be the total number of distinct values in the  $i$ th list (i.e.  $n = \sum_{i=1}^M n_i$ ). For the dataset in Example 1, for example, there are  $n_1=6$  and  $n_2=5$  distinct values in the first and second lists and  $n=11$ . Assume that  $s_j(x)$  is the  $j$ th largest score in the  $i$ th list; then, it is represented by  $n_i$  bits, where the  $(j-1)$  leftmost bits are 0, and the remaining ones 1. Table 9 shows the bitmaps for objects in Example 1. Since  $s_1(x_2)$  is the largest value in the first list, all bits of it are 1. Similarly, since  $s_2(x_2)$  is the 4th largest in the second list, the leftmost  $4-1=3$  bits of its representation are 0, while the remaining ones are 1.

**Table 9.** The bitmap of the dataset in Example 1

Object	Local scores	Bitmap representation
$X_1$	(0.92, 0.87)	(011111, 00011)
$X_2$	(0.95, 0.87)	(111111, 00011)
$X_3$	(0.88, 0.95)	(000111, 11111)
$X_4$	(0.87, 0.90)	(000011, 01111)
$X_5$	(0.89, 0.85)	(001111, 00001)
$X_6$	(0.86, 0.88)	(000001, 00111)

Consider now that we want to know the degree of domination of an object, e.g.  $x_5$  with bitmap representation (001111, 00001). The leftmost bits whose value is 1 are the 3th and the 5th, in the first and second lists, respectively. The algorithm creates two bit-strings,  $c_1(x_5) = 110010$  and  $c_2(x_5) = 111111$ , by juxtaposing the corresponding bits (i.e. 3th and the 5th) of every object. Let  $A(x_5) = c_1(x_5) \& c_2(x_5) = 110010$ . The rightmost bits whose value is 0 of  $x_5$  are the 2th and the 4th, in the first and second lists, respectively. Then, the algorithm creates another two bit-strings,  $d_1(x_5) = 110000$  and  $d_2(x_5) = 111101$ , by juxtaposing the corresponding bits (i.e. 2th and the 4th) of every object. Let  $B(x_5) = c_1(x_5) | c_2(x_5) = 111101$ . The 1's in the result of  $C(x_5) = A(x_5) \& B(x_5) = 110000$ , indicate the objects that dominate  $x_5$ , i.e.  $x_1$  and  $x_2$ . Obviously, the degree of domination of any object  $x$  equates the number of 1's in the result of  $C(x)$ . Both the time cost and extra spaces needed of bitmap are  $O(MN^2)$ .

In our experiments, we use brute-force algorithm to fulfill pre-computations because it's easy to implement.

## 4 Experiments

Our algorithms are implemented in C++. We perform our experiments on a Duo T2130 1.86GHz PC with 1GB of memory. We use both synthetic and real dataset to evaluate NRA, DNRA and ADNRA algorithms. The metrics we measure is the number of sorted accesses performed since the middleware cost is the number of sorted accesses performed times the cost per sorted access.

### 4.1 Description of Datasets

We do experiments on five synthetic datasets and one real dataset. All generated local scores belong to the interval  $[0, 1]$ . The five synthetic datasets are produced to model different input scenarios; they are UI, NI, EI, CO and AC, respectively. UI, CO and AC are generated using the same methodology as [5]. UI contains datasets where object's local scores are uniformly and independently generated for the different lists. NI contains datasets where object's local scores are normally and independently generated for the different lists. EI contains datasets where object's local scores are exponentially and independently generated for the different lists. CO contains datasets where object's local scores are correlated. In other words, the local score  $s_i(x)$  of an object  $x$  is very close to  $s_j(x)$  with high probability, where  $i \neq j$ . To generate an object  $x$ , first, a number  $u_x$  from 0 to 1 is selected using a Gaussian distribution centered at 0.5.  $x$ 's local scores are then generated by a Gaussian distribution centered at  $u_x$  with variance 0.01. Finally, AC contains datasets where object's local scores are anti-correlated. In this case, objects that are good in one list are bad in one or all other lists. To generate an object  $x$ , first, we pick a number  $u_x$  from 0 to 1, like we did for CO datasets. This time, however, we use a very small variance, so that  $u_x$  for different  $x$  are very close to 0.5 and to each other. The local scores of  $x$  are then generated uniformly and normalized to sum up to  $u_x$ . In this way, the overall scores of all objects are quite similar, but their individual scores vary significantly.

**Table 10.** Default settings of experimental parameters

Parameter	Default values
Number of objects, i.e. $N$	100,000
Number of lists, i.e. $M$	5
$k$	20
$K$	20
Aggregate function	summation

For synthetic datasets, our default settings for different parameters are shown in Table 10. In our tests, the default number of data items in each list is 100,000, i.e.  $N=100,000$ . Typically, users are interested in a small number of top answers, thus we set  $k = 20$  as  $k$ 's default value. DNRA needs to pre-compute the  $K$ -skyband and unless otherwise specified we set  $K = 20$  since we set  $k = 20$ . Like many previous works on

top- $k$  query processing, such as [12], we choose the aggregate function as the sum of the local scores. In half of our tests, the number of lists, i.e.  $M$ , is a varying parameter. When  $M$  is a constant, we set it to 5 since most top- $k$  algorithms are evaluated on the dataset with no more than 5 lists.

For real dataset, as did in [12], we choose El Nino dataset (<http://kdd.ics.uci.edu>), which contains oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific. The data is expected to aid in the understanding and prediction of El Nino/Southern Oscillation (ENSO) cycles. We neglect the objects missing some fields. The remaining dataset contains 93935 objects. We chose 7 lists to test our algorithms. And we normalize the dataset with the formula:  $\frac{s_i(x) - Min}{Max - Min}$ , where  $s_i(x)$  is  $x$ 's  $i$ th local score.

## 4.2 Experimental Results

### 4.2.1 Effect of the Number of Lists

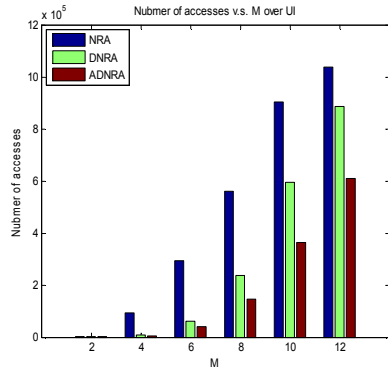
In this section, we compare the performance of our algorithms with NRA over the datasets described in section 4.1 while varying the number of lists.

Over the datasets we considered, with the number of lists increasing up to 12 and the other parameters set as in Table 10, Fig.5-10 show the results measuring the number of sorted accesses performed. The results show us that both our algorithms perform several orders of magnitude fewer sorted accesses than NRA. The reason is that our algorithms already get some information about the objects through pre-computation and they only access the potential objects that can be top- $k$  answers. As  $M$  becomes small, the advantage of our algorithms over NRA increases. The reason for this increase is that the  $K$ -skyband contains fewer objects as  $M$  becomes smaller. Over UI and CO, the ratio of the number of sorted accesses performed by ADNRA to that performed by DNRA is nearly 2/3. And the ratio nearly decreases to 1/2 over NI, EI and AC. Over El Nino Data, although the ratio is a little larger, ADNRA still performs significantly fewer accesses than DNRA. The reason why ADNRA outperforms DNRA is that ADNRA classifies the objects into  $N$  categories while DNRA only classifies objects into two categories. After ADNRA accesses  $D_0, D_1, \dots, D_i$ , the value of  $t$  is already relatively big, which makes it perform fewer accesses on  $D_{i+1}$ .

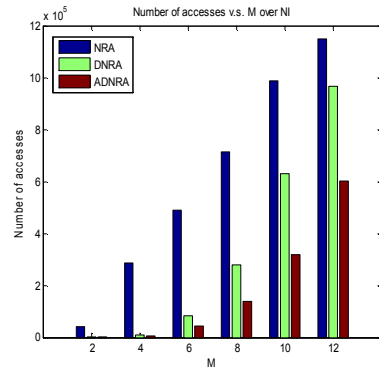
### 4.2.2 Effect of $k$

In this section, we study the effect of  $k$ , i.e. the number of top objects requested, on performance. Since the number of top objects requested varies with different users, we need to set the parameter  $K$  in pre-computation of DNRA as a sufficiently large const, i.e. the upper bound of the number of top objects requested. In our experiments below, we set  $K=100$ , and the other parameters as in Table 9 except that  $k$  varies from 10 to 100.

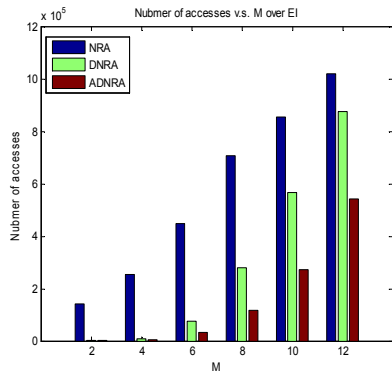
Fig.11-16 show the results of our experiments over the considered datasets. The results show that both our algorithms perform orders of magnitude fewer accesses than NRA. For DNRA and ADNRA, the number of their accesses both increase slightly with  $k$  over all the considered datasets. But the situation is different for NRA.



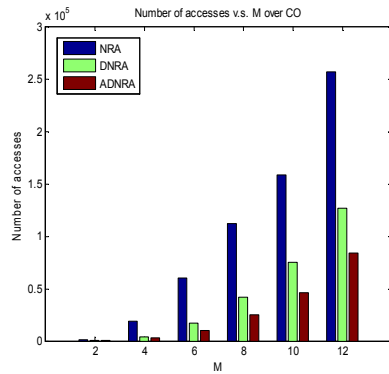
**Fig.5.** The number of accesses v.s. the number of lists over UI,  $k=20$



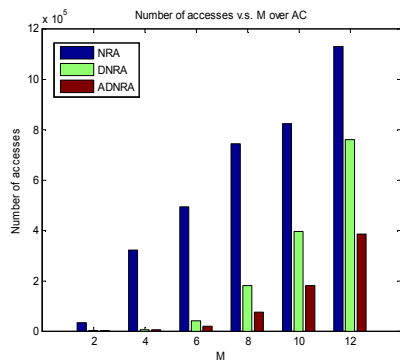
**Fig.6.** The number of accesses v.s. the number of lists over NI,  $k=20$



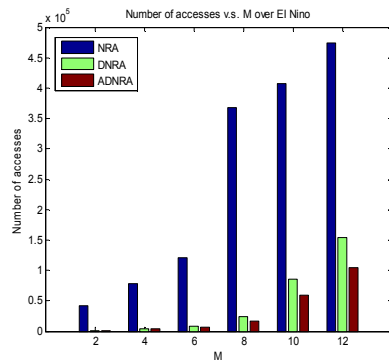
**Fig.4.** The number of accesses v.s. the number of lists over EI,  $k=20$



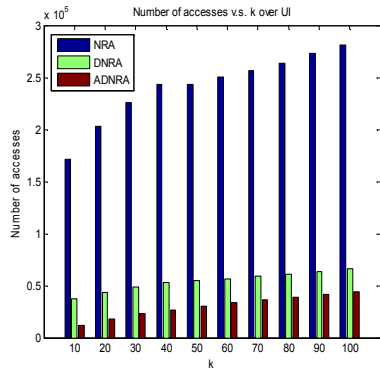
**Fig.8.** The number of accesses v.s. the number of lists over CO,  $k=20$



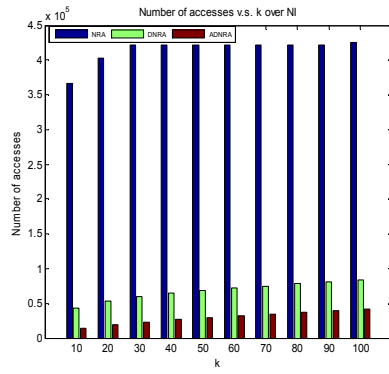
**Fig.9.** The number of accesses v.s. the number of lists over AC,  $k=20$



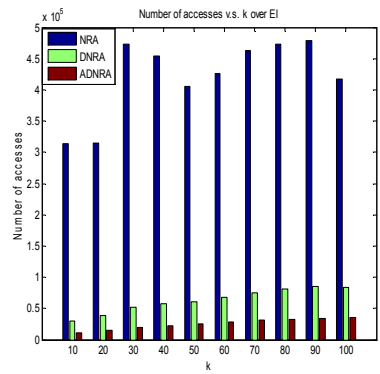
**Fig.10.** The number of accesses v.s. the number of lists over El Nino Data,  $k=20$



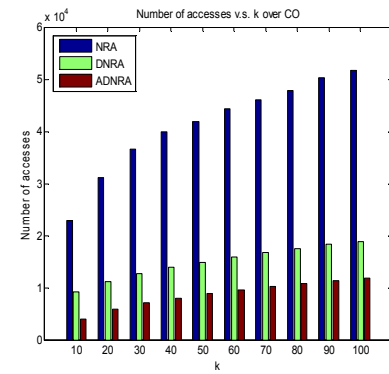
**Fig.11.** The number of accesses v.s.  $k$  over UI,  $M=5$



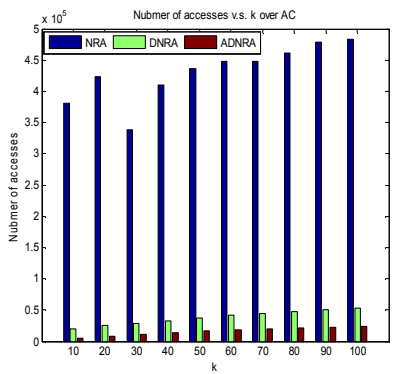
**Fig.12.** The number of accesses v.s.  $k$  over NI,  $M=5$



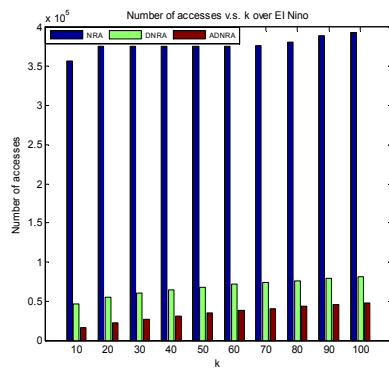
**Fig.13.** The number of accesses v.s.  $k$  over EI,  $M=5$



**Fig.14** The number of accesses v.s.  $k$  over CO,  $M=5$



**Fig.15.** The number of accesses v.s.  $k$  over AC,  $M=5$



**Fig.16.** The number of accesses v.s.  $k$  over EI Nino Data,  $M=5$

The number of accesses performed by NRA increases with  $k$  over UI and CO, almost does not increase over NI and EI Nino data and even fluctuates over EI and AC. The predominance of ADNRA over DNRA becomes more apparent as  $k$  becomes smaller. The reason is that when  $k$  is small, i.e.  $k < K$ , DNRA may perform some useless accesses. Specifically, DNRA may still access those objects whose degree of domination is no less than  $k$ , but those objects can not be top- $k$  answers for any increasingly monotone aggregate function. However, ADNRA will never access those objects.

### 4.3 Summary

Our experiments illustrate that DNRA and ADNRA do several orders of magnitude fewer sorted accesses than NRA, both on synthetic datasets and real dataset. The reason is that our algorithms classify objects and reduce the accessing scope. As  $M$  becomes smaller, the decrease of sorted accesses becomes more significant. The reason is that the  $K$ -skyband contains fewer objects as  $M$  becomes smaller. Algorithm ADNRA does much fewer sorted accesses than algorithm DNRA. For some instances, ADNRA performs 50% fewer accesses than DNRA. This could be explained by the fact that DNRA only classifies objects into two categories depending on whether they belong to the  $K$ -skyband or not while ADNRA takes a further step to classify the objects into  $N$  categories according to their degree of dominations.

## 5 Related Works

Efficient processing of top- $k$  queries is both an important and hard problem that is still receiving much attention. *Skyline* (skyline is a special instance of  $K$ -skyband, where  $K=1$ ) computation is another kind of queries that is also attracting much attention.

There are intrinsic relations between the two kinds of queries. Some papers have tried to use these kinds of relations to address top- $k$  queries, such as [15] and [16]. In [15], A.Vlachou et al. proposed a framework called SPEERTO to support top- $k$  query processing over horizontally partitioned data stored on peers organized in a super-peer network. The key idea of SPEERTO is that each peer computes its  $K$ -skyband as a pre-processing step and each super-peer maintains and aggregates the  $K$ -skyband sets of its peers to answer any incoming top- $k$  query. Essentially, SPEERTO classifies objects into two categories depending on whether they belong to the  $K$ -skyband or not. Our work makes a further step to classify objects into  $N$  categories according to their degree of dominations.

In [16], L.Zou and L.Chen proposed an indexing structure, called DG (Dominant Graph). Based on that, they proposed *Traveler* algorithms to answer top- $k$  queries. Our work is different from [16] mainly in two aspects. Firstly, the models of dataset we use are different. The dataset in [16] consists of one list with  $N$  data items which are like  $(x, s_1(x), s_2(x), \dots, s_M(x))$ , where  $x$  is an object,  $s_i(x)$  is  $x$ 's  $i$ th local score. And each data item can be random accessed. However, the dataset in our paper consists of  $M$  lists with  $N$  data items, which is more general. And each data item can be only

sorted accessed. Secondly, both our works are based on the classifications of the objects, but our methods to classify objects are different. DG uses *Maximal layer* [17] to classify objects while we use degree of domination to classify objects (*Note*: the first Maximal layer contains the same objects as sub-dataset  $D_0$ , which are the skyline objects of the dataset).

We did not compare our algorithms with *Traveler* algorithms directly because they have different methods to access data item, i.e. *Traveler* algorithms use random access to access data item while we only use sorted access. Actually, authors in [16] compared *Traveler* algorithms with TA and CA [1], but they did not compare them with NRA. Certainly, *Traveler* algorithms can also be used in cases where data item can only be sorted accessed and our approaches can also be generalized to cases where random access is possible.

## 6 Conclusions and Future Works

In this paper, we propose DNRA and ADNRA algorithms to address top- $k$  queries, both of which are based on the partition of the dataset. DNRA classifies the objects into two categories depending on whether they belong to the  $K$ -skyband or not. And it only accesses the  $K$ -skyband objects when answering top- $k$  queries. Furthermore, ADNRA classifies the objects into  $N$  categories according to their *degree of domination*. And it only accesses  $k$  categories in order to find the top- $k$  objects. Experimental comparisons show that our algorithms perform several orders of magnitude fewer sorted accesses than NRA and that ADNRA performs significantly fewer sorted accesses than DNRA on some datasets. The reason why our algorithms outperform NRA is that they already know some information about the objects through pre-computation and they only access those potential objects that can be the real top- $k$ . DNRA is powerful in top- $k$  system where  $k$  is a constant. ADNRA can be used in any top- $k$  system where the aggregate function is increasingly monotone and  $k$  is a constant or not.

As  $M$  becomes larger,  $D_0$  and  $D_1$  contain more and more objects, which results in our algorithm ADNRA performs more and more accesses. In the future, we plan to propose some kind of method to continue classifying objects in  $D_0$  and  $D_1$  in order to make ADNRA stop earlier. Furthermore, the variation of ADNRA using dataset-level-parallel policy is fit for parallel environments, so we will optimize it by parallel computing in our future work. Specifically, we first distribute the sub-datasets  $D_0, D_1, \dots, D_{k-1}$  onto  $k$  processors; secondly, let the  $i$ th processor computes in parallel the top- $(k-i)$  objects in  $D_i$ , and store them in  $Y_i$ ; thirdly, we combine  $Y_0, Y_1, \dots, Y_{k-1}$  to get the final top- $k$  answerers.

## 7 Acknowledgement

This work is supported by the National Science Foundation of China under the grant No.60533020 and No.60873210. This work is also supported by the Science Research

Fund of MOE-Microsoft Key Laboratory of Multimedia Computing and Communication (Grant No. 06120801).

## References

1. R. Fagin, A. Lotem and M. Naor.: Optimal aggregation algorithms for middleware. In: *PODS Conf.*, 2001.
2. U.Guntzer, W.-T.Balke and W.KieSling.: Towards efficient multi-feature queries in heterogeneous environments. In: *IEEE Int'l Conf. on Information Technology (ITCC)*, 2001.
3. I.F.Ilyas, W.G.Aref and A.K.Elmagarmid.: Joining ranked inputs in practice. In: *VLDB Conf.*, 2002.
4. I.F.Ilyas, W.G.Aref and A.K.Elmagarmid.: Supporting top-k join queries in relational databases. In: *VLDB Conf.*, 2003.
5. S.B'orzs'onyi, D.Kossmann and K.Stocker.: The skyline operator. In: *ICDE Conf.*, 2001.
6. W.-T. Balke, W. Nejdl, W. Siberski and U. Thaden.: Progressive distributed top-k retrieval in peer-to-peer networks. In: *ICDE Conf.*, 2005.
7. B.Kimelfeld and Y.Sagiv.: Finding and approximating top-k answers in keyword proximity search. In: *PODS Conf.*, 2006.
8. B.Babcock and C.Olston.: Distributed top-k monitoring. In: *SIGMOD Conf.*, 2003.
9. P.Cao and Z.Wang.: Efficient top-k query calculation in distributed networks. In: *PODC Conf.*, 2004.
10. R.Akbarinia, E.Pacitti and P.Valduriez.: Reducing network traffic in unstructured P2P systems using Top-k queries. In: *Distributed and Parallel Databases 19(2)*, 2006.
11. R. Akbarinia, E. Pacitti and P. Valduriez.: Processing top-k queries in distributed hash tables. In: *Euro-Par Conf.*, 2007.
12. J.Yuan, G.Z.Sun, Y.Tian, G.L.Chen and Z.Liu.: Selective-NRA Algorithms for Top-k Queries. In: *APWeb/ WAIM Conf.*, 2009
13. K. Tan, P. Eng and B. Ooi.: Efficient Progressive Skyline Computation. In : *VLDB Conf.*, 2001.
14. D.Papadias, Y.Tao, G.Fu and B.Seeger.: Progressive skyline computation in database systems. In: *ACM, Transactions on Database Systems*, 30(1):41–82, 2005.
15. A.Vlachou, C.Doulkeridis, K.Norvag, M.Vazirgiannis.: On Efficient Top-k Query Processing in Highly Distributed Environments. In: *SIGMOD Conf.*, 2008
16. L.Zou and L.Chen.: Dominant Graph: An Efficient Indexing Structure to Answer Top-K Queries. In: *ICDE Conf.*, 2008
17. T.H.Cormen, C.E.Leiserson, R.L.Rivest and C.Stein *Introduction to algorithms*. The MIT Press, 2001