

# Scalable Ranking for Preference Queries\*

Ying Feng   Divyakant Agrawal   Amr El Abbadi   Ambuj Singh

Department of Computer Science  
University of California, Santa Barbara  
{yingf, agrawal, amr, ambuj}@cs.ucsb.edu

## ABSTRACT

Top- $k$  preference queries with multiple attributes are critical for decision-making applications. Previous research has concentrated on improving the computational efficiency mainly by using novel index structures and search strategies. Since current applications need to scale to terabytes of data and thousands of users, performance of such systems is strongly impacted by the amount of available memory. This paper proposes a scalable approach for memory-bounded top- $k$  query processing.

## Categories and Subject Descriptors

H.2.4 [Database Management]: Systems—*Query Processing*

## General Terms

Performance, Algorithm

## Keywords

Preference queries, Query Optimization

## 1. INTRODUCTION

There has been increasing interest in top- $k$  preference queries from information retrieval (IR) and decision-making applications. In these applications, the data sets usually consist of data entities with a variety of attributes which can be used to order the data. The data entities are objects from the application domains such as images or documents. The attributes which can be used to rank objects are called *rank attributes*. For example, hotels could be ordered according to such attributes as price, number of rooms, and quality of service. Users might need to rank the data entities with different metrics represented as a function defined over a selection of rank attributes. The metrics are referred to as *score functions*. Usually the score functions are *monotonically* increasing or decreasing w.r.t the rank attributes.

Top- $k$  query processing is usually studied in two scenarios: in a distributed systems model, a.k.a middleware systems, or in database systems. The solutions of top- $k$  query [4, 5, 7] in middleware systems is applicable, but neither scalable nor efficient when used in database systems. For example, in our experiments with TA (Threshold Algorithm), for a uniform dataset with 10,000 objects, when the independent

\*This work was supported by NSF under the following grants: CNF-04-23336, IIS-02-23022 and IIS-02-09112

rank attributes grow from 2 to 8, the number of objects accessed using the TA increases from 400 to 8,000, which is 80% of the size of the entire dataset.

Most research on top- $k$  queries in database systems [2, 3, 9] improves performance by utilizing customized index structures and efficient search strategies on the indexes. The best-first search algorithm [8] can be shown to be optimal in terms of the number of node accesses, when applied to top- $k$  queries using multi-dimensional indexes. At each iteration, the node with the highest best possible score is chosen as the next one to be expanded till  $k$  data tuples are generated.

Memory plays an important role in query performance. Although the best-first search algorithm ensures optimality, the size of the priority queue, that is, the memory requirement grows with the number of nodes intersecting with the optimal search range, thus proportional to the data set size. Therefore, the available memory might not always meet the memory needs of the proposed query processing methods, given the increasing data amount especially in web-based applications. Furthermore, the client-base demanding top- $k$  preference queries is expanding rapidly. As many as 150,000 queries need to be serviced in peak time and hundreds of concurrent users send requests. We, therefore, propose a scalable approach to ensure both flexible memory bounds and computational optimality.

## 2. DOMINANCE RANK

The analysis above suggests to bound the data space in order to bound the memory usage in the optimal best-first search algorithm. This objective is consistent with the optimization based on the cost model for minimizing object accesses in Fagin's algorithms [4, 5] for distributed systems. Fagin's first algorithm (FA) [4] can be applied to generate the reduced candidate data set  $FA(k)$  for top- $k$  queries with arbitrary monotonic score functions. The idea is to sort tuples by their rank attribute values and access sorted lists of tuples simultaneously, just as scanning sorted lists of objects in FA. The candidate data include the tuples retrieved when  $k$  common tuples are accessed from all sorted rank attributes. However, the candidate set can be further minimized by exploiting the *dominance rank* of  $d$ -dimensional data sets.

**DEFINITION 1.** Let  $t$  denote a data tuple for a given  $d$ -dimensional data set, the *dominance rank* of  $t$ ,  $dr(t)$ , is the number of data tuples dominating the tuple  $t$ .

Intuitively, the dominance rank of point  $p$  is the number of data points dominating point  $p$ . For example, tuple  $s_1$  has dominance rank 0 (Figure 1). Based on the *dominance rank*, we can further introduce the dominance rank  $k$  set  $DR(k)$  as  $\{t | dr(t) < k\}$ . For the example of Figure 1,  $DR(2) = DR(1)$

| A normalized relation |    |    |    |                |
|-----------------------|----|----|----|----------------|
|                       | r1 | r2 | r3 | dominance rank |
| S1                    | 9  | 9  | 5  | 0              |
| S2                    | 7  | 10 | 6  | 0              |
| S3                    | 6  | 8  | 4  | 2              |
| S4                    | 8  | 5  | 7  | 0              |

DR(1)=DR(2)={S1, S2, S4}  
DR(3) = {S1, S2, S3, S4}

Figure 1: Dominance ranks of an example table

$= \{s_1, s_2, s_4\}$ . It is straightforward to see from the definition that  $DR(k) \subseteq DR(k+1)$ . Also as expected,  $DR(k)$  can be proven as the minimal candidate data set for top- $k$  queries with any arbitrary monotonic score functions. Moreover, it can be shown that  $FA(k) \supseteq DR(k)$ .

### 3. SCALABLE TOP-K QUERY PROCESS

Dominance rank can reduce the data space for any given top- $k$  query by partitioning data a priori. Then top- $k$  query evaluation can be decomposed on multiple partitions.

#### 3.1 Partitioning with dominance rank

The preprocessing phrase includes three steps: (1) to compute the dominance rank of the whole data set; (2) to partition data tuples; (3) to build a multi-dimensional index on each partition. As the third step is based on indexes of choice, we detail the first two steps below.

The **computation of dominance rank** can be accomplished by a multidimensional divide-and-conquer algorithm for all-point ECDF rank problem proposed by Bentley [1]. The computation complexity is  $O(N \log^{d-1} N)$  for any  $d > 1$ , and  $d$  is the number of rank attributes and  $N$  is the number of data tuples. An alternative to achieve better preprocessing time at the expense of approximation is to approximate  $DR(k)$  using  $FA(k)$ . Therefore, the approximation algorithm costs linear time complexity,  $O(Nd)$ .

The next step, **partitioning the data space**, aims to bound the partition size preferably by some constants. Therefore, the *threshold size partitioning* is employed here. It divides the whole datasets into  $p$  ordered subsets:  $DR(k_1), DR(k_2) - DR(k_1), \dots, DR(k_p) - DR(k_{p-1})$ , where  $p$  is the number of partitions and  $k_1 < k_2 < \dots < k_p$ . The partitioning proceeds to ensure bounded partition size using  $O(n)$  time complexity: First, the data set is scanned in the ascending order of dominance ranks. The data points with dominance rank  $k$  are put into a bucket  $bin[i]$  until the  $bin[i]$  size exceeds the max size, threshold  $\tau$ . Then a new bucket  $bin[i+1]$  is created to accommodate data points with dominance rank  $k+1$ . Figure 2(b) shows an example for partitions with threshold 200, i.e. each partition size is at least 200.

Theoretically, if  $\tau > \max_i |DR(i) - DR(i-1)|$ , that is, threshold  $\tau$  is greater than any set of dominance rank  $k$  data, the partition size is bounded within  $\tau$  and  $2\tau$ .

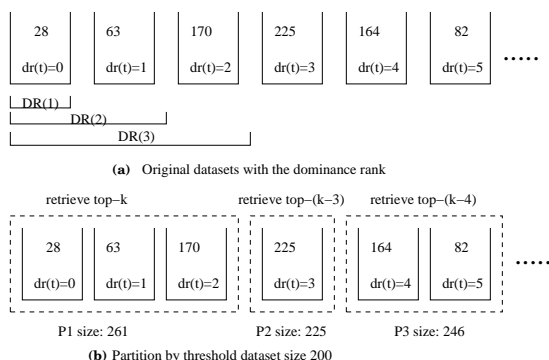
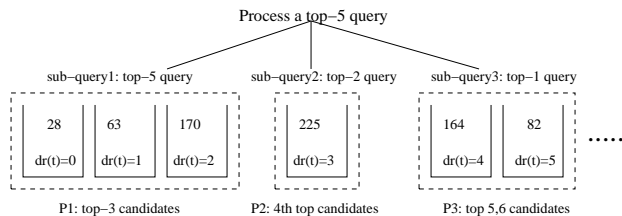


Figure 2: Reducing the search space by partitions



(1) generate top-3 answers; => (2) generate 4-th answer; => (3) generate 5-th answer

Sequential sub-query processing

Figure 3: Top- $k$  query on partitions

### 3.2 Decomposed Top- $k$ Queries over Partitions

In order to process top- $k$  preference queries over partitioned data, two questions need to be addressed: (1) Which partitions need to be examined to answer the top- $k$  preference queries correctly? (2) What kind of queries should be posted on those partitions?

The first question can be answered by the results in Section 2. Only the partitions containing data from set  $DR(k)$  is possible to be a top- $k$  answer for any monotonic score function. As an example in Figure 3, a top-5 query involves three partitions:  $P_1, P_2, P_3$ .

For each of the partitions to be examined, a top- $k$  query can be decomposed into several top- $k$  sub-queries on them. The following lemma answers the second question: how the top- $k$  queries are decomposed.

LEMMA 1. *Suppose the dominance ranks of data in Partition  $P_i$  range from  $k_{i-1} + 1$  to  $k_i$ , where  $k_{i-1} < k$ . Partition  $P_i$  can only contribute to at most  $k - k_{i-1}$  members in any top- $k$  query result set, ranging from the  $(k_{i-1} + 1)$ -th to  $k$ -th answers. Therefore, the top- $k$  query can be decomposed to a top- $(k - k_{i-1})$  sub-query on each partition  $P_i$  containing the candidate answers.*

For example, in Figure 3, a top-5 query is decomposed into a top-5 query on  $P_1$ , a top-2 query on  $P_2$  and a top-1 query on  $P_3$ . The algorithm to process sub-queries is not described here due to space limit [6].

## 4. CONCLUSION

We address the challenges of memory bounds using optimal best-first search algorithm to improve the scalability for preference queries in database systems. The advantages of the proposed technique include: (1) It provides a scalable approach to process top- $k$  preference query with flexible memory bounds; (2) The query processing is progressive.

## 5. REFERENCES

- [1] J. L. Bentley. Multidimensional divider-and-conquer. *Communications of the ACM*, pages 214–229, 1980.
- [2] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS*, 27(2):153–187, 2002.
- [3] Y. Chang, L. Bergman, V. Castelli, C. Li, M. L. Lo, and J. Smith. The onion technique: Indexing for linear optimization queries. In *SIGMOD*, pages 391–402, 2000.
- [4] R. Fagin. Fuzzy queries in multimedia database systems. In *PODS*, pages 1–10, 1998.
- [5] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
- [6] Y. Feng, D. Agrawal, A. E. Abbadi, and A. Singh. Scalable and optimal ranking for preference queries. Technical report, UCSB, 2005.
- [7] U. Guntzer, W. Balke, and W. Kiebling. Optimizing multi-feature queries for image database. In *VLDB*, 2000.
- [8] G. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD*, pages 83–95, 1995.
- [9] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, 2003.