HKU CS Tech Report TR-2011-06

Efficient All Top-*k* Computation

A unified solution for all top-*k*, reverse top-*k* and top-*m* influential queries

Shen Ge [#1] Leong Hou U [*2] Nikos Mamoulis [#3] David W.L. Cheung [#4]

[#]Department of Computer Science, University of Hong Kong
Pokfulam Road, Hong Kong
[1]sge@cs.hku.hk [3]nikos@cs.hku.hk [4]dcheung@cs.hku.hk

[*]Department of Computer and Information Science, University of Macau
Av. Padre Tomás Pereira Taipa, Macau
[2]ryanlhu@umac.mo

Abstract

Given a set of objects *P* and a set of ranking functions *F* over *P*, an interesting problem is to compute the top ranked objects for all functions. Evaluation of multiple top-*k* queries finds application in systems, where there is a heavy workload of ranking queries (e.g., online search engines and product recommendation systems). The simple solution of evaluating the top-*k* queries one-by-one does not scale well; instead, the system can make use of the fact that similar queries share common results to accelerate search. This paper is the first, to our knowledge, thorough study of this problem. We propose methods that compute all top-k queries in batch. Our first solution applies the block indexed nested loops paradigm, while our second technique is a view-based threshold algorithm. We propose appropriate optimization techniques for the two approaches and demonstrate experimentally that the second approach is consistently the best. In addition, we demonstrate the utility of our approach in solving other complex queries that depend on the computation of multiple top-*k* queries. We show that our adapted methods for these complex queries outperform the state-of-the-art by orders of magnitude.

1 Introduction

Many real life applications support ranking of products according to user preference functions. For example, assume that a customer desires to purchase blu-ray discs from an online store (e.g., Amazon). The store would rank the blu-ray discs according to the preferences of the customer; these preferences could be expressed by the user explicitly, or they could be implicitly derived from the customer purchase records [1]. For instance, assume that *movie cast* and *release date* are the two features of blu-ray discs for which customers mostly care. Recent movies having a good cast would rank higher than others. To simplify illustration and analysis, we assume that these features take values from a normalized numerical domain; e.g., the quality of casting takes a score from 0 (worst) to 1 (best). This way, the products can be modeled by multidimensional points; e.g., points p_1, p_2, and p_3 are used to represent three products respectively in Fig. 1. Modeling objects in such a multidimensional feature space is common for diverse types of queries, such as top-k queries [10, 12, 20], skyline queries [8, 19], and market analysis queries [23, 24].

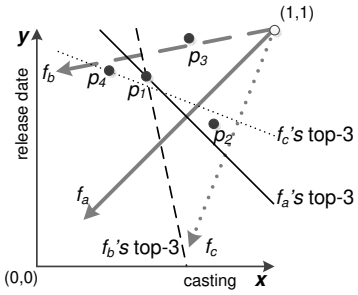
Given a preference function f, we can rank the products $p \in P$ according to $f(p)$. Fig. 1 shows three linear functions f_a, f_b, and f_c which create three object rankings as shown in the right part of the figure. Each function is of the form $f[x]x + f[y]y$, such that $0 \leq f[x], f[y] \leq 1$ and $f[x] + f[y] = 1$. The functions are represented as vectors in the space that contains the points. The object ranking for a specific function f can be determined by the order of the points are met if we sweep a line perpendicular to the vector of f from point $(1, 1)$ towards point $(0, 0)$. In general, different customers may have completely different preferences. For instance, f_b represents the preferences of a customer, u_b who is concerned much

more about the quality of casting than the release date. Accordingly, p_2 is the best product according to u_b's preferences.

Generally speaking, users are more interested in top ranked products. Given a constant k, top-k queries[10, 12, 20] can be defined in Definition 1, according to a given ranking function f.

Definition 1 (Top-k query, $TOP^k(f)$)**.** *Given a set of products P, a preference function f, and a positive integer k, the top-k query $TOP^k(f)$ returns a subset of k products from P, such that $f(p_i) \geq f(p_j)$, $\forall p_i \in TOP^k(f), p_j \in P \backslash TOP^k(f)$.*

For example, in Fig. 1, there are four products in the system and only two features, quality of casting and release date, are taken into account. Suppose that $k = 3$ and there is a customer u_a who equally weighs these two features. As shown in the figure, u_a's preferences are captured by a linear function $f_a = 0.5x + 0.5y$; this corresponds to the order of the points are swept by the perpendicular plane to a vector from $(1, 1)$ to $(0, 0)$. According to f_a, the system recommends $TOP^3(f_a) = \{p_3, p_2, p_1\}$ to u_a.



	f_a ——	f_b - - -	f_c ·····
top-1 disc	p_3	p_2	p_3
top-2 disc	p_2	p_3	p_1
top-3 disc	p_3	p_1	p_4

Figure 1: Top-k queries for online stores

In most online recommendation applications, top-k queries for multiple users may have to be evaluated simultaneously. Recommendation systems of online stores is such an application. As an example, consider a second-hand cars company wants to recommend cars to customers before the summer season; the company issues multiple top-k queries, one for each customer (depending on his/her individual preferences), simultaneously. The customer preferences may have to be collected from previous purchase records of the users. The recommendations can be computed by issuing an individual top-k query for each user, $TOP^k(f_i)$. This iterative approach becomes too expensive when a large number of queries have to be evaluated over a large number of products. Thus, developing specialized techniques for processing multiple top-k queries is an important problem that has been overlooked in past research. Definition 2 formally defines this problem.

Definition 2 (All top-k query, $ATOP^k$)**.** *Given a set of products P, a set of preference functions F, and a positive integer k, the all top-k query $ATOP^k$ returns $TOP^k(f)$ for every function $f \in F$.*

To the best of our knowledge, there is no efficient approach to compute multiple top-k queries simultaneously ($ATOP^k$). In this paper, we study two *batch processing* techniques for this problem. The first is a *batch indexed nested loops* approach and the second is a *views-based threshold* algorithm. We also propose several novel optimization techniques for these methods.

Besides products recommendation, other tasks, such as product promotion analysis[21] and identifying the most influential products [22], can benefit from an efficient approach for computing multiple top-k queries simultaneously, as we discuss in Section 2. We demonstrate the utility of our result in these complex analysis tasks; when $ATOP^k$ is used as search module for reverse top-k [21] and top-m influential [22] queries, the evaluation cost of these queries greatly decreases.

In the rest of the paper, we assume that object features (i.e., dimensions) are normalized in the range $[0, 1]$, and larger values are better. In addition, we assume that the set F of multiple ranking queries contains only linear preference functions and the coefficients of every function are normalized; the score $f(p)$ of an object p is computed by the inner product $\sum_{i=1}^{d} f[i] \cdot p[i]$ of f's weights vector with p's feature vector.

The rest of the paper is organized as follows. The applicability of all top-k queries is discussed in Section 2. An intuitive batch processing technique is introduced in Section 3. In Section 4, we present an alternative batch processing approach which extends the views-based threshold algorithm [11] and fully optimize it. Section 5 discusses how we can extend our techniques to support related queries, including reverse top-k and top-m influential queries. In Section 6, we experimentally evaluate our methods using synthetic and real data. Section 7 discusses related work. Finally, Section 8 concludes the paper. Table 1 summarizes the notation used throughout the paper.

Table 1: Summary of Notations

Symbol	Meaning
F	the set of user preferences
P	the set of products
$f(p)$	score of product p by user preference f
$p[i]$	the i-th dimension value of p
$f[i]$	the i-th coordinate (weight) of f
TOP^k	a top-k query
$ATOP^k$	a all top-k query
$RTOP^k$	a reverse top-k query
$I^k(p)$	top-k influence score of product p
$ITOP_k^m$	a top-m influential query

2 Applications

Product promotion is one possible application that can make use of $ATOP^k$. Suppose that a property agent is promoting a new building to customers via web advertisements. To minimize cost, the agent should advertise the building only to those customers who are potentially interested in it; in other words, product p_i should be advertised to users who highly rank p_i, based on their known preferences. This problem is known as *reverse top-k* ($RTOP^k(p_i)$) query [21], which is formally defined in Definition 3. Intuitively, given product p_i and a set of user preferences, $RTOP^k(p_i)$ returns the users who have p_i in their top-k result.

Definition 3 (Reverse top-k query[21])**.** *Given a product p, a positive integer k, a set of products P and a set of user preferences F, the reverse top-k query $RTOP^k(p)$ returns a subset of user preferences F, such that $RTOP^k(p) \subseteq F$, and $f_i \in RTOP^k(p)$ if and only if $\exists q \in TOP^k(f)$ such that $f(p) \geq f(q)$.*

For example, for the data in Fig. 1, $RTOP^2(p_2)$ returns functions f_a and f_b since p_2 is ranked 2nd and 1st by f_a and f_b, respectively. We note that the solution for this problem proposed in [21] does not scale well, because every reverse top-k query is answered by issuing a set of essential top-k queries. If multiple reverse top-k queries are issued (e.g., multiple products are to be promoted at a holiday season), some of these top-k queries might even have to be executed multiple times.

A related application is to find products of *significant impact* in the market. Identifying products of high influence in a large database (e.g., database of houses, second-hand cars, etc.) is an important market analysis task, which can help companies to assess the popularity of their current products and/or design new ones with features similar to most popular products. For instance, the iPad is considered a good product because it is ranked highly by many customers in a survey [2]. Intuitively, the influence of a product in the market is the number of customers who consider it intriguing (i.e., rank it high in their preferences). In Fig. 1, p_3 is the most intriguing product since it is ranked highly by all three functions. The problem of finding the most influential products has been recently studied by Vlachou et al. [22]. They define the

influence score $I^k(p_i)$ of a product p_i in Definition 4, using the number of customers have p_i in their top-k preferences.

Definition 4 (Influence score, I^k[22])**.** *Given product dataset P, user preferences F and a positive integer k, the influence score of a product p is defined as $I^k(p) = |F'|$, where $F' \subseteq F$ and $F' = RTOP^k(p)$.*

Accordingly, the top-m most influential query is defined in Definition 5. The ranking criterion is based on the influence scores I^k.

Definition 5 (Top-m influential query[22])**.** *Given a product dataset P, a set of users preferences F and a positive integer k. The top-m influential query $ITOP_k^m$ returns a subset of m products from P, such that $ITOP_k^m \subseteq P$ and $|ITOP_k^m| = m$, $I^k(p_i) \geq I^k(p_j)$, $\forall p_i \in ITOP_k^m$, $p_j \in P \backslash ITOP_k^m$.*

For example, in Fig. 1, let $k = 3$ and consider the three user preference functions $F = \{f_a, f_b, f_c\}$. The four products $\{p_1, p_2, p_3, p_4\}$ have influence scores $\{3, 2, 3, 1\}$, respectively. The score of p_4 is only 1 because it appears in the top-3 set of only one function (f_c). Thus, $ITOP_3^2$ returns $\{p_3, p_1\}$. In [22], the object influence scores are calculated by reverse top-k queries, therefore the proposed solution does not scale well according to our discussion above.

In Fig. 2, we briefly summarize the relationship between the all top-k ($ATOP^k$) query that we study in this paper and $RTOP^k(f)/ITOP_k^m$. In [21], a reverse top-k query $RTOP^k(f)$ is computed by a set of top-k queries; however, not all these queries need to be evaluated due to the use of pruning strategies. In addition, according to [22], the influence score of a product $I^k(p)$ is equivalent to the size of the reverse top-k result. Given a set of products and a set of preference functions, the top-m influential query $ITOP_k^m$ is evaluated using the influence scores of the products. Therefore, a large number of top-k queries are implicitly involved in a top-m influential query. Although pruning strategies and fine-tuned execution ordering are employed in the state-of-the-art solutions for $RTOP^k(f)$ and $ITOP_k^m$ queries in [21] and [22], respectively, neither solution optimizes the core $ATOP^k$ module of these queries. In other words, an efficient evaluation technique for all top-k queries ($ATOP^k$) would greatly benefit the evaluation of $RTOP^k(f)/ITOP_k^m$ queries.

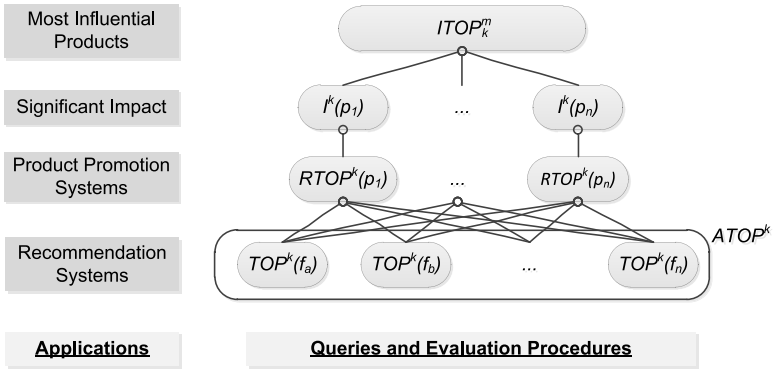


Figure 2: Relationship of different queries

3 Batch Top-k Processing

Top-k queries are extensively studied in the literature [10, 11, 12, 20]. The state-of-the-art techniques aim at minimizing the cost of a *single* top-k query with the use of thresholding and/or indexing structures. However, there is a lack of research on multiple top-k evaluation. Motivated by this, in this section, we propose a batch processing technique that indexes not only the objects but also the functions, to support all top-k computation.

Roughly speaking, batch processing can be considered as the counterpart of block indexed nested-loops in relational databases and spatial join queries in spatial databases [26]. Suppose that the objects are

indexed by a multidimensional index, e.g., R*-tree [6], and the functions are also partitioned in groups. To group the functions, we can first order them according to their position on the Hilbert curve [7] that indexes the space of function coefficients. Then, we split the curve into subintervals, each defining a group, such that each group contains no more than a ratio δ of the functions. Intuitively, a group contains a small number of *similar* functions that would share a number of results. Processing the functions in the group simultaneously would be faster than executing the queries individually, as some search cost would be shared among the functions in the group. In Section 6, we study the choice of δ.

Let F_g be a group of functions; the group maximum score $s_{max}^{F_g}(p)$ of an object p computed by the functions of the group is $s_{max}^{F_g}(p) = \sum_{i=1}^{d} \max_{f_g \in F_g}\{f_g[i]\}p[i]$. For a given F_g, we traverse the nodes and objects in the R*-tree (e.g. Fig. 3(a)) in descending order of the group maximum score. We first load the root of the R*-tree, calculating $s_{max}^{F_g}$ of all entries in it (i.e., for each minimum boundary rectangle (MBR)). The maximum possible score $s_{max}^{F_g}(m)$ of an MBR m is the maximum score of any possible object inside m. If higher values are preferred in each dimension, the corner point of an MBR corresponding to the combination of the largest values is the point with the maximum score. We put all accessed R*-tree entries and their maximum scores into a priority queue and access them in descending maximum score order. Each time an entry e is de-heaped, if e is a non-leaf entry (e.g., M_a in Fig. 3(a)), we calculate the maximum scores for all its children and insert them into the priority queue. If e is a leaf MBR (e.g., m_b in Fig. 3(a)), then all functions in F_g are computed using the points in the corresponding leaf node. As an optimization (see Lemma 1 below), we avoid processing an MBR m for a function $f \in F$ if the upper bound $f(m)$ (computed using the best corner of m) is worse than the k best scores of f computed so far. We name this batch processing technique as Batch Indexed Nested Loops algorithm (BINL). We list the pseudocode for BINL in Algorithm 1.

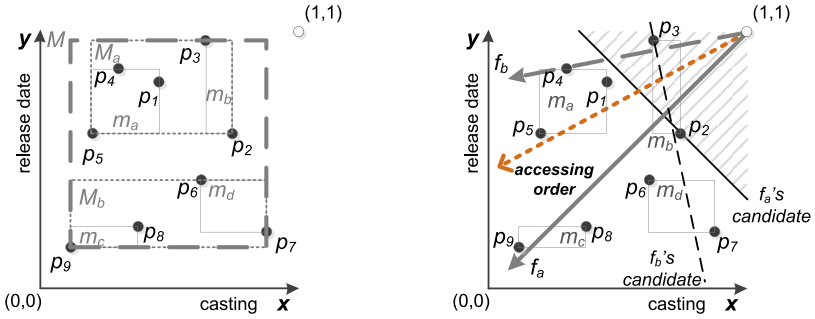
Algorithm 1 BINL Algorithm

Algorithm $BINL(R, F, k)$
R is the R*-Tree index of the set of objects P
1: partition F into a set of g groups $\{F_1, \ldots, F_g\}$ by Hilbert curve
2: **for all** $F_i \in \{F_1, \ldots, F_g\}$ **do**
3: en-heap $(R.root, 0)$ into PQ
4: **while** PQ is not empty **do**
5: de-heap the top element m from PQ
6: **if** m is an non-leaf MBR **then**
7: **for all** $m_i \in m$ **do**
8: compute the maximum possible score $s_{max}^{F_i}(m_i)$ to m_i
9: en-heap $(m_i, s_{max}^{F_i}(m_i))$ into PQ
10: **else if** m is a leaf MBR **then**
11: **for all** $f_i \in F_i$ **do**
12: **if** $f_i(m_i)$ is better than k-th candidate of f_i **then**
13: evaluate f_i for all objects in m_i

Lemma 1 (MBR Pruning)**.** *An MBR m needs not be evaluated by a function f if $f(m)$ is no better than the k-th score for the objects being seen so far, where $f(m)$ is the maximum score of function f for any point in m.*

Fig. 3(b) illustrates an example for *BINL*. Assume that we are processing the group of functions $F_g = \{f_a, f_b\}$. The accessing order based on $s_{max}^{F_g}$ can be conceptually captured by the order a perpendicular plane to the dashed arrow in the figure crosses the MBRs. Suppose that $k = 2$ and we have already accessed four MBRs, M, M_a, m_b, and M_b. Then, p_2 and p_3 have been seen by f_a and f_b already and we only have $\{m_d, m_a, m_c\}$ in the priority queue. Next, we get m_d from the priority queue, which is a leaf MBR, therefore its contents are evaluated using the functions in F_g. Note that only f_b evaluates the objects in m_d while f_a prunes m_d because $f_a(m_d) < f_a(p_2) < f_a(p_3)$.

Discussion. Techniques similar to BINL have been proposed before for All Nearest Neighbors Queries



(a) R*-tree grouping

(b) BINL processing

Figure 3: An example of batch indexed nested loops

(ANN) [26] in spatial databases. We note that BINL does not support early termination, because the group traversing order is generally different from the early termination order of every single user preference function in that group, which means that we have to traverse the R*-tree once for every functions group.

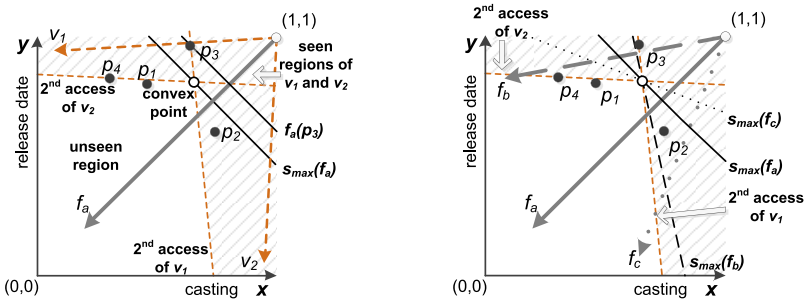
4 A View-based Approach

In this section, we investigate an alternative, more efficient approach than BINL. A well-accepted general paradigm for efficient query processing, for different data and query types, is to take advantage of materialized views with pre-computed results [13]. Answering top-k queries using materialized views has been studied in [11]. The materialized top-k views used could be cached results of previously computed top-k queries. The algorithm proposed in [11] is a Linear Programming adaptation of the Threshold Algorithm (LPTA). The execution paradigm is based on the Threshold Algorithm (TA) [12]: an iterative processing technique which combines two or more ranked object lists. LPTA sequentially accesses the results of two or more materialized rankings of objects, based on different functions, in order to compute the top-k objects based on a new function. When an object p is accessed from view v_i, a random access is performed at each of the other views to calculate the aggregate feature score of object p. LPTA keeps track of the k objects with the highest scores seen so far. These k objects will become the final top-k result if they have better scores than the *maximum possible score* for all unseen objects. The maximum possible score is computed by *linear programming* in [11].

We demonstrate LPTA by an example in Fig. 4(a). In this example, we use the same objects set from Fig. 1 and construct two views, v_1 and v_2. We can find the top-1 result of f_a by sorted accesses to these views using LPTA. Assuming that v_1 and v_2 have been accessed 2 times, the regions being accessed are shaded in the figure. Note that the unseen region must be convex if all preference functions are linear. Given a linear function, the maximum score of any objects in the convex unseen region must be smaller than or equal to the scores of the convex points (of the unseen region), which can be computed by linear programming. After these sorted accesses, only three objects, p_2, p_3, and p_4, are seen so far and the preference function f_a keeps p_3 as the top-1 candidate. LPTA returns p_3 for f_a since the current maximum possible score $s_{max}(f_a)$ is already worse than the candidate's score, $f_a(p_3)$.

To support batch processing, when an object p is accessed from a view, we can evaluate its scores for multiple top-k queries. For every top-k query being evaluated, we update the current result set if necessary. A function is marked as *stopped* if its k-th candidate score is no worse than the maximum possible score. Based on this idea, we can answer multiple top-k queries by traversing each view once. We call this method Batched Linear Programming adaptation of the Threshold Algorithm (BLPTA). The pseudo code of BLPTA can be found in Algorithm 2.

At every iteration of BLPTA, we fetch the next object p from one of the views in a round-robin fashion and update the top-k candidates for each of the *running* functions. In Fig. 4(b), the top-1 candidates of f_a, f_b, and f_c are p_3, p_2, and p_3 respectively after the 2^{nd} access from each of views. The *maximum possible scores*, s_{max}, of the functions are illustrated by three different lines in Fig. 4(b). In this example,



(a) LPTA computation

(b) BLPTA computation

Figure 4: Top-k computation using ranking views

Algorithm 2 BLPTA Algorithm

Algorithm $BLPTA(V, P, F, k)$
1: **for all** $f \in F$ **do**
2: $TOP^k(f) \leftarrow \emptyset$ and mark f as *running*
3: **while** F is not empty **do**
4: **for all** $v \in V$ **do** ▷ access the views in round-robin fashion.
5: fetch next object p from v and update accessed regions
6: **for all** $f \in F$ not marked as *stopped* **do**
7: **if** $f(p)$ is better than k-th object $TOP^k(f)$ **then**
8: remove k-th object and insert p into $TOP^k(f)$
9: compute maximum possible score $s_{max}(f)$
10: **if** k-th object in $TOP^k(f)$ is better than $s_{max}(f)$ **then**
11: mark f as *stopped* and remove f from F

all functions are marked as *stopped* after the 2^{nd} access from each of views. Therefore, BLPTA exits the iterative process and returns the all top-k result.

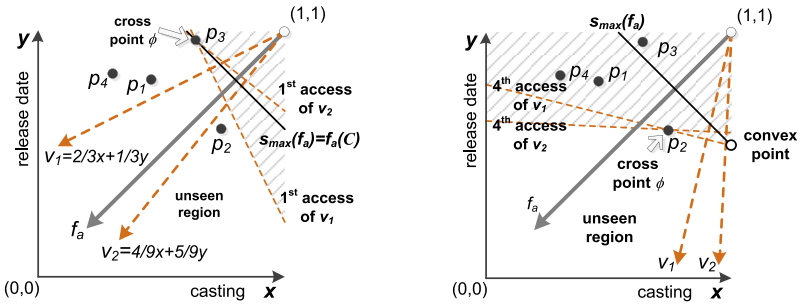
BLPTA can terminate early if all functions are marked as *stopped*. However, BLPTA can be slow since (1) the maximum possible scores are computed by linear programming, (2) functions are not partitioned into groups, and (3) every object being accessed from views is unavoidably evaluated. In remaining of this section, we discuss and resolve these three issues and propose an optimized version of the algorithm, called Efficient adaptation of the Threshold Algorithm (ETA).

4.1 Avoiding linear programming

The maximum possible score in BLPTA is computed by linear programming. Considering the fact that this computation will be carried out for all *running* preference function against all *accessed* objects, it is very time consuming. Motivated by this, we redesign our method as follows. Instead of using previously computed ad-hoc views, before all top-k evaluation, we *construct the views*, using some constraints, such that the maximum possible score can be derived from the cross point of d *hyperplanes* (technique to be discussed shortly). We first introduce the constraints that we impose when constructing views (Definition 6).

Definition 6 (d-bounding views)**.** *A preference function f is bounded by d views $\{v_1, \ldots, v_d\}$ if and only if there exists a d-dimensional vector r, such that $\forall r_i, r_i \geq 0$ and $\sum_{i=1}^{d} r_i v_i = f$.*

Intuitively, a preference function f being bounded by d views means that the direction of f is enclosed by the directions of d views. Fig. 5(a) demonstrates an example of d-bounding views. Suppose that $f_a = \frac{1}{2}x + \frac{1}{2}y$ and consider two views, $v_1 = \frac{2}{3}x + \frac{1}{3}y$ and $v_2 = \frac{4}{9}x + \frac{5}{9}y$, in the system. There exists a vector $r = (\frac{1}{4}, \frac{3}{4})$ that makes $r_1 v_1 + r_2 v_2 = f_a$. Therefore, we say that views v_1 and v_2 are a set of d-bounding views for f_a.



(a) By d-bounding views (b) By other views

Figure 5: Example of different views settings

Besides, we define as the *scanning hyperplane* of a view v, the hyperplane which is perpendicular to the v's vector and intersects the last object seen in v. The dashed lines (orthogonal to the preferences vectors) in Fig. 5 illustrate scanning hyperplanes. Formally, if s is the last score seen in v, the scanning hyperplane of v is defined by the set of points x which satisfy $v[1]x[1] + \ldots + v[d]x[d] = s$.

In BLPTA (and LPTA), the maximum possible score s_{max} is computed by linear programming which is a well accepted method for linear optimization problems. Consider a preference function f, d views $(v_1, \ldots, v_d)$, and their last accessed scores s_i. The optimization problem can be defined as the follow.

$$\begin{aligned} &\text{maximize} && f(x) \\ &\text{subject to} && v_i(x) \geq s_i,\ i = 1, \ldots, d. \end{aligned}$$

However, it is not necessary to execute linear programming if x can be determined simply. By basic geometry, we can easily show that there is only one cross point ϕ being intersected by d hyperplanes in the d dimensional space. We illustrate the cross point ϕ in Fig. 5(a). Assume that all user preferences in the system are bounded by d views. Theorem 1 shows that the cross point ϕ is the point x that maximizes the score of any unseen objects (proofs of all theorems are in Appendix A). For completeness, we show in Fig. 5(b) that if f is not bounded by the views, then $f(\phi)$ is no longer the maximum possible score.

Theorem 1. *For a set of user preferences F being bounded by d-bounding views $(v_1, \ldots, v_d)$, $f(\phi)$ is no worse than the score of any unseen objects, where ϕ is the cross point of the scanning hyperplanes of the d-bounding views.*

According to Theorem 1, $f(\phi)$ is equivalent to the maximum possible score $s_{max}(f)$ in BLPTA. Clearly, we can mark a function as *stopped* if f is bounded by the corresponding d-bounding views and the value of $f(\phi)$ is not better than the k-th candidate score. The remaining problem is to calculate the cross point ϕ of d scanning hyperplanes. This can be done by solving a simple linear system. For every view v_i and its last seen score s_i, we have

$$v_i[1]\phi[1] + \ldots + v_i[d]\phi[d] = s_i$$

Since we have d different equations in total, ϕ can be found by solving a simple linear system, $\phi = A^{-1}B$, where A is the set of d views and B is the set of last seen scores. Formally:

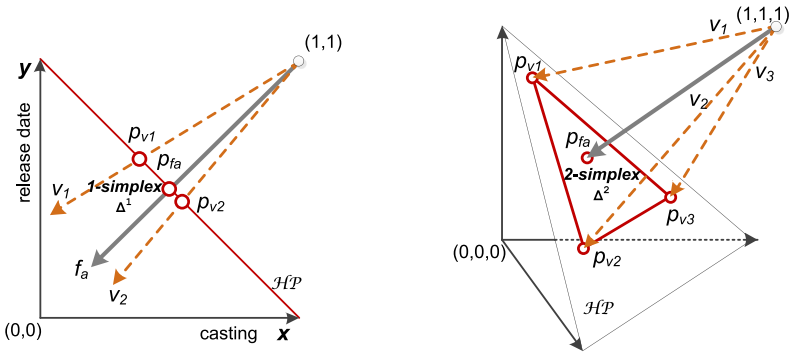
$$\phi = \begin{pmatrix} v_1[1] & \ldots & v_1[d] \\ v_2[1] & \ldots & v_2[d] \\ \vdots & \vdots & \vdots \\ v_d[1] & \ldots & v_d[d] \end{pmatrix}^{-1} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_d \end{pmatrix}$$

Discussion. The views based computation can stop early if the preferences functions are bounded *tightly* by the views. For instance, we can mark f_a as *stopped* after accessing one object from each of views in Fig. 5(a); while we need to access four objects in total from the views in Fig. 4(a). However, finding the

tightest d-bounding views is a hard problem in high dimensional spaces. The most loose d-bounding views are the base views (e.g., $v_1 = x$, $v_2 = y$, and $v_3 = z$ in the 3D space). In next section, we study how to tighten these views by a partitioning technique.

4.2 Views-based partitioning

We can take advantage of partitioning the functions into groups instead of processing them one-by-one. Before we introduce the partitioning process, we show how to construct a $(d-1)$-simplex by projecting the vectors of d-bounding views to a hyperplane $\mathcal{HP}$ (i.e., $\mathcal{HP}(\mathcal{X}) = x[1] + \ldots + x[d] = 1$). For a set of d-bounding views, we can find their corresponding point using a linear system. For instance, p_{v_1} and p_{v_2} are the corresponding points of v_1 and v_2, respectively, in Fig. 6(a). These d corresponding points construct a $(d-1)$-simplex ($\Delta^{d\text{-}1}$) [18] on hyperplane $\mathcal{HP}$, that is a $(d-1)$-dimensional generalization of a 2D triangle or a 3D tetrahedron. In Fig. 6, we illustrate two such simplexes in 2 and 3 dimensional spaces (the 1-simplex Δ^1 is a line segment and the 2-simplex Δ^2 is a 2D triangle).



(a) 1-simplex (b) 2-simplex

Figure 6: Examples of d-bounding views projection

A simplex can easily be partitioned by a point inside it (see Definition 7). In Fig. 7, for example, we have three basic bounding views and four functions in the 3D space. On the hyperplane, we create a Δ^2 according to the corresponding points from v_1, v_2, and v_3. We can partition the Δ^2 into three sub-simplexes (i.e., Δ^2_1, Δ^2_2, and Δ^2_3) by adding the view v_4 (see Fig. 7(b)).

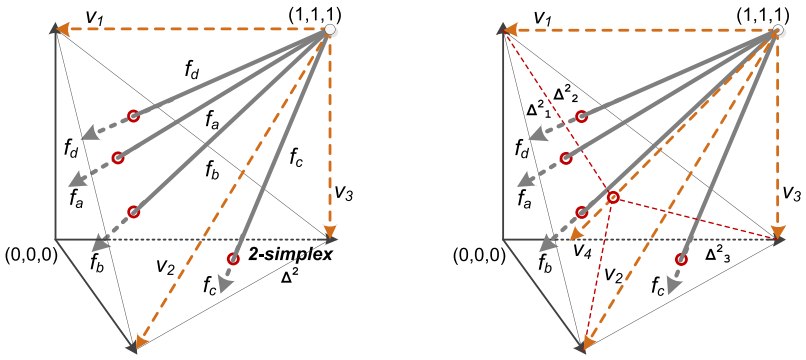
Definition 7 (Simplex partitioning)**.** *Given a $\Delta^{d\text{-}1}$ and a point p inside the simplex, $\Delta^{d\text{-}1}$ can be partitioned into d isolated $\Delta^{d\text{-}1}$s being split from p towards the vertices of the simplex.*

Theorem 2 shows that the function f_a passes through point p_{f_a} in the interior of $\Delta^{d\text{-}1} = \{p_{v_1}, \ldots, p_{v_d}\}$ if and only if f is bounded by $\{v_1, \ldots, v_d\}$. In Fig. 7(b), the corresponding d-bounding views of Δ^2_1, Δ^2_2, and Δ^2_3 are $\{v_1,v_2,v_4\}$, $\{v_1,v_3,v_4\}$, and $\{v_2,v_3,v_4\}$, that bound functions $\{f_a,f_b\}$, $\{f_d\}$, and $\{f_c\}$, respectively.

Theorem 2. *A function (or a view) is bounded by a set of d-bounding views if and only if it passes through the interior of the $(d\text{-}1)$-simplex defined by the d-bounding views.*

Note that simplex partitioning creates new sets of d-bounding views that are tighter than the original d-bounding views. This makes computation more efficient as discussed in Section 4.1. For instance, finding the top-k result of f_d using $\{v_1, v_3, v_4\}$ is faster than using $\{v_1, v_2, v_3\}$. For the sake of generating tighter boundings, we can recursively partition the simplex. On the other hand, this might create a large amount of views. Therefore, there is a tradeoff between achieved tightness and the number of views, which should be considered in the process.

Accordingly, we propose an algorithm that recursively partitions the initial simplex. After each partitioning, we assign each function to the sub-simplex where its projection falls. We use a parameter λ to control the number of views being created during this process. We do not further split a simplex if the



(a) Simplex construction (b) Splitting using v_4

Figure 7: An example of partitioning

number of functions being bounded by it is less than a ratio λ of the total. The partitioning algorithm can be found in Algorithm 3.

In Algorithm 3, we first construct the simplex $\Delta^{d\text{-}1}$ based on the d-bounding views V (e.g., v_1, v_2, and v_3 in Fig. 7) and assign the entire set of preferences functions to $\Delta^{d\text{-}1}.F$, where $\Delta^{d\text{-}1}.F$ denotes the associated preference function set F of the simplex Δ^{d-1}. Lines 3 to 12 describe an iterative process that recursively partitions the simplex. Given a point inside a simplex (e.g., the average point of all vertices, v_{avg}), we can partition the simplex $\Delta^{d\text{-}1}$ into d sub-simplex using Definition 7 (in line 5). Every bounding function f of $\Delta^{d\text{-}1}$ is assigned to one of the d sub-simplexes. Clearly, the simplex is not tight enough if it bounds many functions. Therefore, we further partition a sub-simplex if the number of bounding functions is larger than a threshold (e.g., being controlled by λ in our algorithm).

Algorithm 3 d-bounding views partitioning

Algorithm $partitioning(V, F, \lambda)$
1: construct $\Delta^{d\text{-}1}$ for V and set $\Delta^{d\text{-}1}.F := F$
2: push $\Delta^{d\text{-}1}$ into a queue Q
3: **while** Q is not empty **do**
4: $\quad\Delta^{d\text{-}1} := Q.pop()$
5: $\quad$partition $\Delta^{d\text{-}1}$ into $\{\Delta_1^{d\text{-}1}, \ldots, \Delta_d^{d\text{-}1}\}$ using $v_{avg} :=$ AVG$v_i \in V$
6: $\quad$**for all** $f \in \Delta^{d\text{-}1}.F$ **do**
7: $\quad\quad$assign f to $\Delta_i^{d\text{-}1}$ if f is in the interior of $\Delta_i^{d\text{-}1}$
8: $\quad$**for all** $\Delta_i^{d\text{-}1} \in \{\Delta_1^{d\text{-}1}, \ldots, \Delta_d^{d\text{-}1}\}$ **do**
9: $\quad\quad$**if** $size(\Delta_i^{d\text{-}1}.F) \geq \lambda \cdot size(F)$ **then**
10: $\quad\quad\quad$push $\Delta_i^{d\text{-}1}$ into Q ▷ further partition $\Delta_i^{d\text{-}1}$
11: $\quad\quad$**else**
12: $\quad\quad\quad F_g := F_g \cup \Delta_i^{d\text{-}1}.F$
13: **return** F_g

4.3 Accessing multiple objects from views

Recall that whenever a leaf MBR m is accessed by BINL, every function f_g in F_g first examines whether m can be pruned by the candidate set of f_g, according to Lemma 1 (see Section 3). However, the objects being accessed from views are unavoidably evaluated by the functions in BLPTA. For the sake of batch pruning, we fetch ω objects from a view instead of one object at each access. Subsequently, we construct an MBR for these ω objects and apply the same pruning idea as BINL, such that not every object is necessarily evaluated by the functions, improving the efficiency of pruning.

4.4 Putting all together

We are now ready to present our ETA algorithm (Efficient adaptation of the Threshold Algorithm), which integrates all techniques been discussed. ETA first partitions the functions into groups such that each of group is bounded by a corresponding set of d-bounding views (see Section 4.2). For every group, we evaluate the functions in batch using the corresponding d-bounding views. At every iteration, in each group, we access the views in round-robin fashion. At each access, we fetch ω objects and construct the MBR for these objects (see Section 4.3). Subsequently, we update the cross point ϕ of d scanning hyperplanes (see Section 4.1).

After we construct the MBR m for the accessed objects, we examine whether the objects belonging to m should be examined by a function using the MBR pruning technique (see Lemma 1 in Section 3). Moreover, the result of a function f is confirmed by the condition whether $f(\phi)$ is no better than the candidate set of f, and f is marked as *stopped* in this case (see Section 4.1). The all top-k results of a group found as soon as all functions in the group are marked as *stopped*. Algorithm 4 is a detailed pseudocode for ETA.

Algorithm 4 ETA Algorithm

```
   Algorithm ETA(V, P, F, k, ω, λ)
 1: for all f ∈ F do
 2:     TOP^k(f) ← ∅ and mark f as running
 3: F_g := partitioning(V, F, λ)                          ▷ Section 4.2
 4: for all F ∈ F_g do
 5:     while F is not empty do
 6:         for all v ∈ V do
 7:             fetch next ω object p from v              ▷ Section 4.3
 8:             construct the MBR m for ω objects         ▷ Section 4.3
 9:             compute cross point φ using d-scan hyperplanes   ▷ Section 4.1
10:             for all f ∈ F do
11:                 if f(m) is better than k-th score in TOP^k(f) then
12:                     for all p ∈ m do
13:                         if f(p) is better than k-th score in TOP^k(f) then
14:                             remove k-th object and insert p into TOP^k(f)
15:                 if f(φ) is no better than k-th score in TOP^k(f) then
16:                     mark f as stopped and remove f from F
```

In our implementation for ETA, we assume that the set of objects is indexed by a multidimensional access method and that the views are not pre-computed and materialized. A view is computed on-demand at the simplex partitioning phase (Section 4.2) using an off-the-shelf top-k computation algorithm (e.g., BRS [20]).

5 Efficient reverse top-k and top-m influential computation

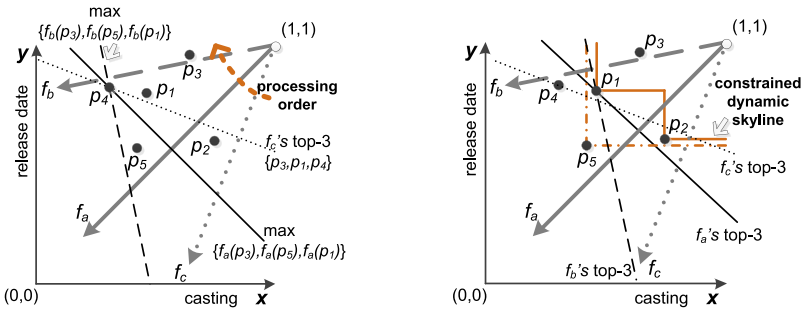
According to our discussion in Section 2, the reverse top-k $RTOP^k(p)$ and top-m influential $ITOP_k^m$ queries can use our all top-k algorithms to accelerate searching. We first briefly review the state-of-the-art solutions to these problems from [21] and [22]. Then, we show how we can evaluate these queries more efficiently using $ATOP^k$.

5.1 State-of-the-art $RTOP^k$ solution

Given a set of objects P and a set of preference functions F, the reverse top-k query of an object $p \in P$ returns the subset of F that contains p in their top-k result. A naïve method computes a reverse top-k query by evaluating the preference functions one by one. [21] proposed the evaluation of the functions in a given order. Intuitively, the top-k results are similar (or exactly the same) if two functions, f_i and f_j, are very

close[1]. In other words, if f_i does not have p in its top-k result, then most probably p is not in f_j's top-k either. Therefore, we can skip the evaluation of f_j if $f_j(p) < \max_{p_i \in TOP^k(f_i)} f_j(p_i)$ since p is ranked worse than at least k other objects. This method is termed Reverse top-k Threshold Algorithm (RTA) in [21]. However, this process might evaluate all functions, in the worst case.

We demonstrate the reverse top-k computation in Fig. 8(a). Given the execution order based on cosine similarity (i.e., f_c, f_a, f_b) and $k = 3$, we want to answer $RTOP^k(p_5)$. According to the given order, we first evaluate f_c where the top-k result is $\{p_3, p_1, p_4\}$ and find that f_c is not in the reverse top-k set of p_5. Before we evaluate next function f_a, we first apply f_a on f_c's top-k set and compute a threshold $\theta = \max\{f_a(p_3), f_a(p_1), f_a(p_4)\})$. In this example, $f_a(p_5) < \theta$, which indicates that f_a is not the reverse top-k of p_5 and needs not be evaluated. On the other hand, $f_b(p_5) \geq \theta$, therefore f_b has to be evaluated.



(a) Reverse top-k query

(b) Top-m influential query

Figure 8: Examples of other queries

5.2 State-of-the-art $ITOP_k^m$ solution

Given a set of objects P, a set of functions F, and k, the top-m influential query returns the m objects that have the highest influence scores, defined by the size of $RTOP^k(p)$. A straightforward solution is to evaluate a reverse top-k query for each object. Note that each reverse top-k query is evaluated by multiple top-k queries. The cost becomes too high if F and P are large. In [22], a technique that estimates the maximum possible influence score $U(q)$ of an object q is proposed. This can be computed by

$$U(q) = |\cap_{\forall p_i \in CDS(q)} RTOP^k(p_i)|,$$

where $CDS(q)$ is the constrained dynamic skyline of q (see Definition 8).

Definition 8 (Constrained Dynamic Skyline Set)**.** *Given a set of objects P and an object q, we denote as $P_c \subseteq P$ the set of all objects p_i, such that $\forall_{i=1}^{d} : q[j] \leq p_i[j]$. An object $p_i \in P_c$ belongs to the constrained dynamic skyline set $CDS(q)$ of object q, if it is not dynamically dominated with respect to q by any other point $p' \in P_c$.*

$CDS(q)$ finds a set of dynamic skyline objects in the region being constrained by q; this region is bounded from q towards the best point $(1, \ldots, 1)$. In the example of Fig. 8(b), suppose k is set to 3, $CDS(p_5)$ contains $\{p_1, p_2\}$ and $U(p_5) = 2$ $(= |\{f_a, f_b\} \cap \{f_a, f_b, f_c\}|)$.

Assuming that P is indexed by a multidimensional access method, we can traverse the objects $p_i \in P$ in decreasing order of $U(p_i)$. Similar to other branch-and-bound (BB) processing techniques (e.g., [20]), the first m de-heaped objects are the result of the query. This BB algorithm is the best approach in [22] and it is much faster than the straightforward solution. However, the top-m influential query essentially executes a large amount of top-k queries indirectly, since every reverse top-k query is evaluated by a set of top-k queries.

[1]Closeness can be measured by a cosine function.

5.3 Using all top-k computation

In this section, we study how we can use $ATOP^k$ to evaluate $RTOP^k$ and $ITOP_k^m$. We also discuss why our approach is superior to the state-of-the-art solutions.

$RTOP^k$ using $ATOP^k$. After having computed an $ATOP^k$, we have the top-k results of all functions. For the objects and functions of Fig. 8, the $ATOP^k$ results are shown in Fig. 9(a). By "inverting" this table, as shown in Fig. 9(b) we can obtain the reverse top-k sets of all objects. Thus, any $RTOP^k$ query can be answered easily by fetching a row in the inverted table. The space requirement is only $O(|F| \cdot k)$.

TOP^k	**top-3 result**
f_a	p_3, p_2, p_1
f_b	p_2, p_3, p_1
f_c	p_3, p_1, p_4

(a) All top-k results

$RTOP^k$	**reverse top-3 result**	I^k
p_1	f_a, f_b, f_c	3
p_2	f_a, f_b	2
p_3	f_a, f_b, f_c	3
p_4	f_c	1

(b) Inverted table

Figure 9: All reverse top-k computation

$ITOP_k^m$ using $ATOP^k$. Having computed the inverted table, which lists the reverse top-k set of each object, we can easily find the influence score of any object by accessing the corresponding row. In fact, for a $ITOP_k^m$ query, we only need the cardinality of each list; our objective is to find the m lists with the largest cardinality. Thus, even if we do not have the inverted table, we can simply scan the all top-k result and find the objects with the largest influence scores. The details are listed in Algorithm 5.

Algorithm 5 Top-m influential query using $ATOP^k$

Algorithm $ITOP - ATOP(V, P, F, k, \lambda)$
1: for $\forall_{p \in P} I^k(p) \leftarrow 0$ ▷ Initialize influence scores
2: $ATOP^k \leftarrow$ run all top-k computation
3: **for all** $f \in F$ **do**
4: **for all** $p \in ATOP^k[f]$ **do** ▷ $ATOP^k[f] \equiv TOP^k(f)$
5: $I^k(p) \leftarrow I^k(p) + 1$
6: **return** the top-m objects p with respect to $I^k(p)$

Discussion. In [21] and [22], many top-k queries are evaluated if F and P are large, while in [22] multiple reverse top-k queries are executed and some of them may even share the same top-k queries, which are evaluated multiple times in this case. For a fair comparison, we implemented an optimized version of BB, named Optimized Branch-and-Bound algorithm (OBB), which caches the results of previously issued top-k queries and reuses them if necessary. Still, as we show in Section 6, OBB is much slower than our "$ITOP_k^m$ using $ATOP^k$" approach.

6 Experimental Evaluation

According to the methodology in [8], we generated three types of datasets, *independent* (IND), *correlated* (COR), *anti-correlated* (ANT). In IND datasets, the feature values are generated uniformly and independently. COR datasets contain objects whose values are correlated in all dimensions. ANT datasets contain objects whose values are good in one dimension and tend to be poor in other dimensions. In addition, we generate *clustered* (CLU) datasets by randomly selecting C independent objects, and treat them as cluster centers. Each cluster object is generated by a Gaussian distribution with mean at the selected cluster center and standard deviation 5% of each dimension domain range. We set C to 10 by default.

In addition, we experimented with two real datasets, NBA [3] and Household [4]. NBA contains 12,278 statistics from regular seasons during 1973-2008, each of which corresponds to the statistics of

an NBA player's performance in 6 aspects (minutes played, points, rebounds, assists, steals, and blocks). Household consists of 3.6M records during 2003-2006, each representing the percentage of an American family's annual expenses on 4 types of expenditures (electricity, water, gas, and property insurance).

All methods were implemented in C++ and the experiments were performed on an Intel Core2Duo 2.66GHz CPU machine with 8 GBytes memory, running on Ubuntu 10.04. Table 2 shows the ranges of the investigated parameters and their default values (in bold). In each experiment, we vary a single parameter, while setting the others to their default values. Our system uses a 4KB page size. In order to measure the exact I/O cost, we assume no memory buffer is available.

Table 2: Range of parameter values

Parameter	Values
$\|P\|$ (in thousand)	10, 25, 50, **100**, 200, 400
$\|F\|$ (in thousand)	10, 25, **50**, 100, 200
Dimensionality d	2, **3**, 4, 5, 6
Data distribution for P	**IND**, ANT, COR, CLU
Data distribution for F	**IND**, CLU
k	2, 5, 10, **20**, 40, 80
m	2, 5, 10, **20**, 40, 80
BINL grouping ratio, δ	0.001, 0.002, 0.005, 0.01, **0.02**, 0.05, 0.1, 0.2, 0.5, 1
ETA splitting ratio, λ	0.001, 0.002, 0.005, 0.01, **0.02**, 0.05, 0.1, 0.2, 0.5, 1
ETA number of accessed objects, ω	1, 2, 5, **10**, 20, 50, 100, 200, 500, 1000

Parameter tuning. We first study the effect of the various tuning parameters on the algorithms, BINL and ETA. We investigate the effect of δ (grouping ratio in BINL), λ (splitting ratio in ETA), and ω (number of accessed objects in ETA). Figure 10(a) shows the effect of δ on the cost of the BINL algorithm for different dimensionality d. For $\delta = 1$ or very small δ, the cost is high since either forming a single group or many small groups is not beneficial for BINL. Therefore we set $\delta = 0.02$ by default; BINL performs well with this value at any dimensionality.

ETA has two parameters, λ and ω, and its cost is affected by both of them. We investigated how various values of these parameters affect the cost. Here, we plot the cost of ETA as a function of one parameter (λ or ω) while setting the other to the default value. Based on the result, we choose $\lambda = 0.02$ and $\omega = 10$ that show robust performance at any dimensionality.

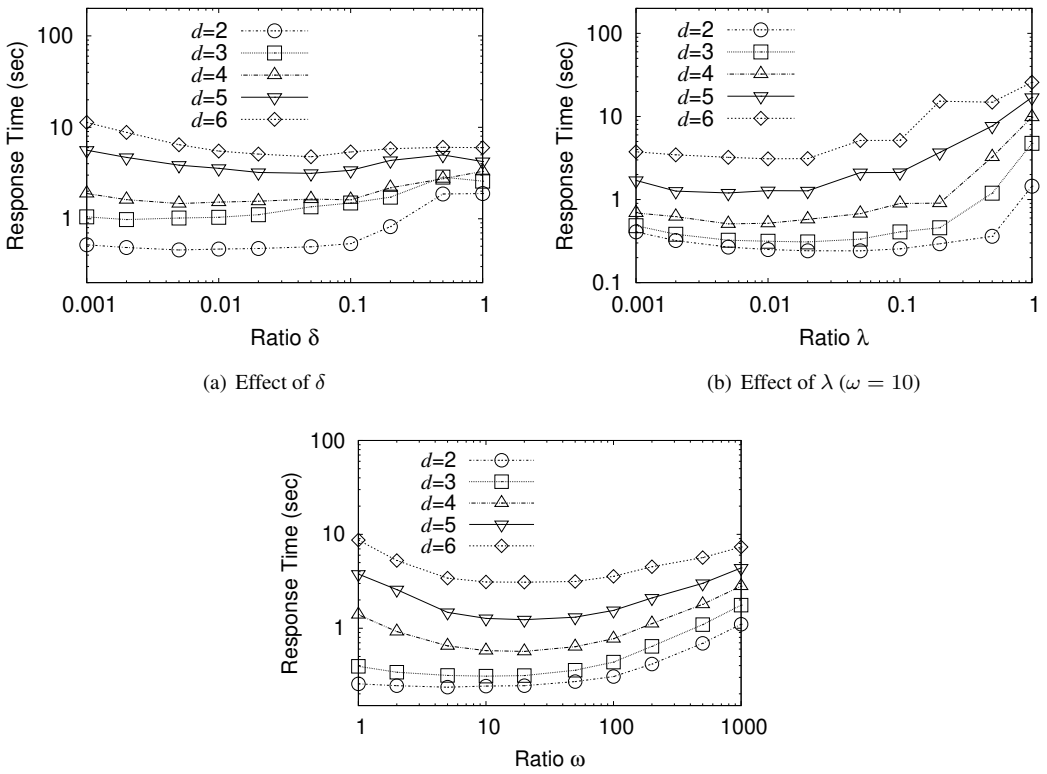
Scalability experiments. In this set of experiments, we demonstrate the superiority of our all top-k methods, BINL (Section 3) and ETA (Section 4.4) to the naïve approach. The naïve approach evaluates the top-k queries one-by-one using BRS [20]. Fig. 11(a) shows the response times of the three methods as a function of dimensionality d, after setting all other parameters to their default values. Cost grows exponentially with d for all methods. ETA is at least 8.7 and 1.6 times faster than Naïve and BINL, respectively in all experiments. For large values of d, the gap between BINL and ETA becomes smaller, since the MBRs that group multiple accessed objects in ETA becomes too large, reducing the effect of the MBR pruning technique.

Fig. 11(b) compares performance as a function of k. ETA is at least 8 and 2.4 times faster than Naïve and BINL, respectively. All methods are sensitive to k since the problem becomes harder as k increases.

The response times for different numbers of products $|P|$ are shown in Fig. 11(c). The cost is not very sensitive to $|P|$ since the products are indexed and we only need to access a small fraction of the data.

Fig. 11(d) shows the response time of all methods for different numbers of functions $|F|$. The response time increases linearly with $|F|$, since there are more top-k queries being evaluated.

Data Distribution. As shown in Fig. 12(a), ETA is at least one order of magnitude faster than Naïve and 2.5 times faster than BINL for different data distributions of P and independently distributed F. ANT



(a) Effect of δ

(b) Effect of λ ($\omega = 10$)

(c) Effect of ω ($\lambda = 0.02$)

Figure 10: Sensitivity experiments

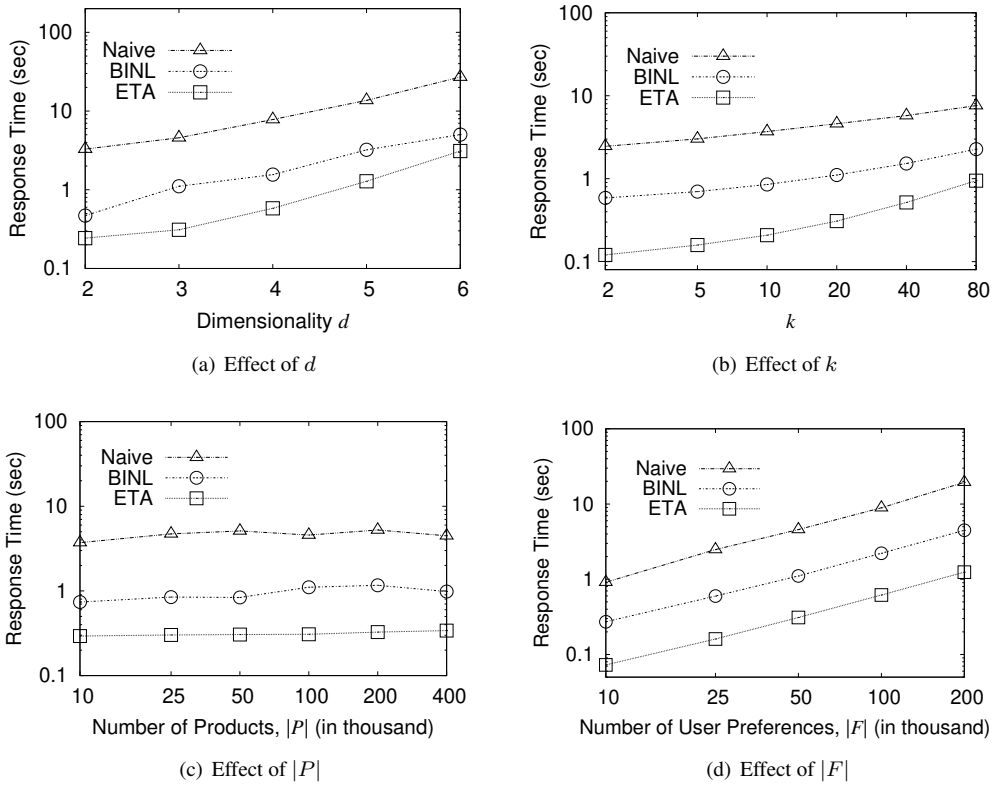
distributed objects are the hardest case since top-k computation becomes hardest in this case. Interestingly, the gap between ETA and the other methods widens in this case. One of the reasons is that our d-bounding views partitioning technique provides better grouping than the Hilbert curve grouping. We also evaluated our methods for the CLU F where we generate the functions coefficients in clusters. As shown in Fig. 12(b), ETA is again the best method which is at least one order of magnitude faster than Naïve and 1.6 times faster than BINL. We conclude that ETA is the best method for all distribution combinations.

Fig. 12(c) plots the response time of all methods on the NBA real dataset. We instantiated P from this dataset (12,278 records) and set other parameters to their default values. Again, ETA is consistently better than Naïve and BINL for all values of d. Summing up, ETA is the best solution for $ATOP^k$ queries, typically being one order of magnitude faster than Naïve solution and 2-3 times faster than BINL.

In Fig. 12(d), we demonstrate the response time of all methods using another real dataset, Household. We instantiated P from the Household dataset (including 3.6M records). We divided Household into four datasets with 516K, 514K, 1.25M, and 1.35M records from years 2003, 2004, 2005, and 2006 respectively. The feature values in Household are discrete, so there are some tuples having the same feature values in all dimensions; in this case the objects are grouped to a single capacitated object. The number of different discrete objects are 242K, 250K, 520K, and 542K, respectively in the four years. We demonstrate the response time of all three methods as a function of the data in these four years in Figures 12(d). ETA again performs best in all cases, being at least 36 and 4 times faster than Naïve and BINL, respectively.

I/O cost and peak memory usage. Fig. 13(a) and 13(b) show the I/O cost and peak memory usage[2] of all three methods as a function of dimensionality d, after setting all other parameters to their default values. The I/O costs of all three methods (Naïve, BINL, and ETA) grow exponentially with the dimensionality. This result is consistent with the corresponding response time experiment (Fig. 11(a)); ETA accesses

[2]We get the peak memory usage by adding/substracting the memory usage of data structures on their construction/destruction.



(a) Effect of d

(b) Effect of k

(c) Effect of $|P|$

(d) Effect of $|F|$

Figure 11: Effect of different parameters for $ATOP^k$

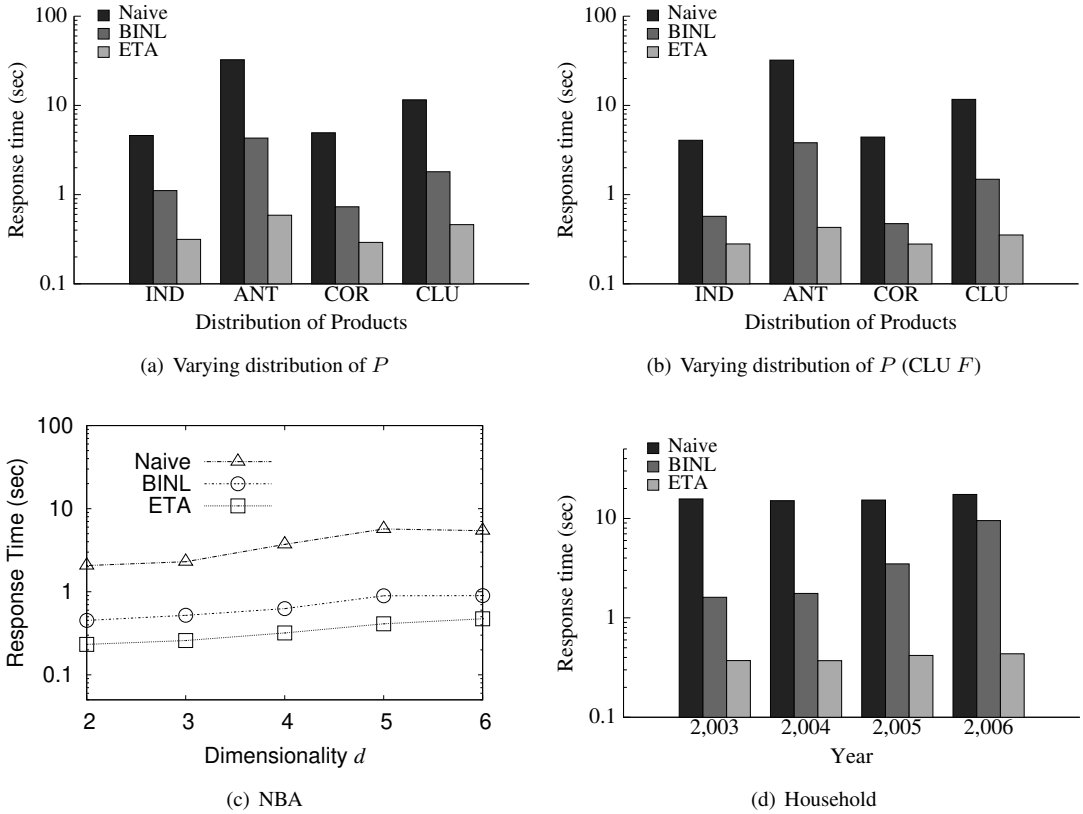
several times to two order of magnitude fewer pages than other two methods. However, ETA may use more memory than Naïve and BINL since each view keeps some data structures for incremental top-k computation; still the required memory is not excessive.

Reverse top-k and top-m influential computation. We now demonstrate the use of $ATOP^k$ queries in the computation of reverse top-k and top-m influential queries. For these two problems, we compare the state-of-the-art solutions [21, 22] to the $ATOP^k$-based alternatives that we introduced in Section 5.3.

For reverse top-k queries, we plot the cost ratio between an $ATOP^k$ query using ETA and a single $RTOP^k$ query using RTA [21]. As shown in Fig. 14(a), RTA is only 1.95 to 13.2 times faster than ETA when dimensionality d varies from 2 to 6. However, ETA computes the all top-k result which can be used to answer *any* reverse top-k result (see Fig. 9(b)). In other words, if we are to execute more than 13 $RTOP^k$ queries in $d = 6$, ETA should be preferred to RTA. Thus, RTA is not appropriate in settings where multiple reverse top-k queries are to be executed. Comparing the two queries for different values of k (Fig. 14(b)) leads to similar conclusions.

For top-m influential queries, we compare our $ITOP_k^m$ using $ATOP^k$ (ITOP-ATOP) approach (see Section 5.3) to the state-of-the-art solution BB and its optimized version OBB (as discussed in Section 5.3). Fig. 15(a) shows the response time for these methods as a function of k. As k increases, OBB becomes much better than original BB since OBB caches the results of previous top-k computations. However, OBB is still 16 times slower than our ITOP-ATOP approach, which performs an all top-k query and uses its results to evaluate the $ITOP_k^m$ query. Fig. 15(b) shows how the cost is affected by m. The response times of BB and OBB are linearly increasing with m, because BB and OBB unavoidably compute more maximum possible influence scores when m becomes larger and this introduces additional reverse top-k queries. However, our approach is completely insensitive to m since we have already collected all necessary data for $ITOP_k^m$ by an $ATOP^k$ computation.

Fig. 16 shows some additional experiments on $ITOP_m^k$ queries (varying dimensionality and data dis-



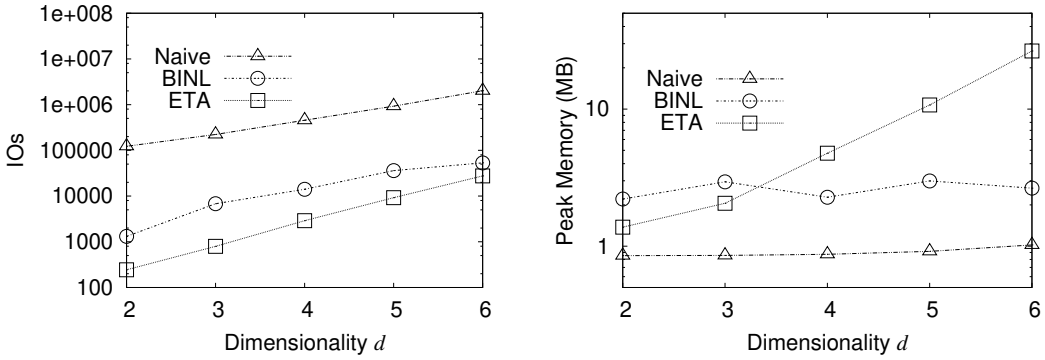
(a) Varying distribution of P

(b) Varying distribution of P (CLU F)

(c) NBA

(d) Household

Figure 12: Varying data distribution



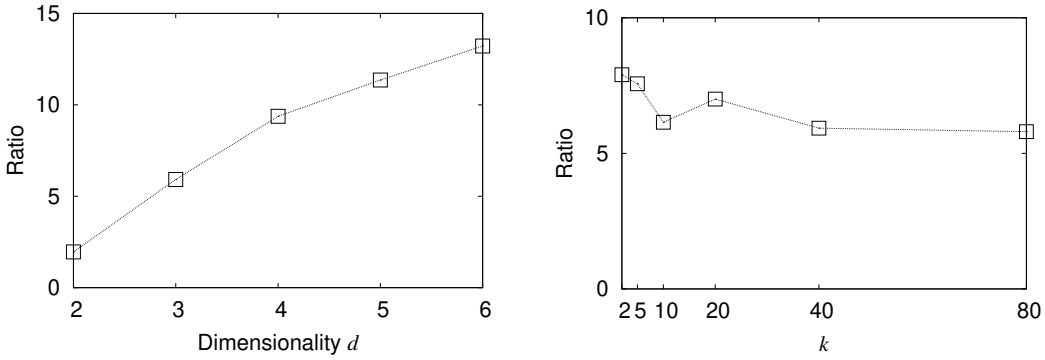
(a) I/O cost vs. d

(b) Peak memory vs. d

Figure 13: Extra experiments for $ATOP^k$

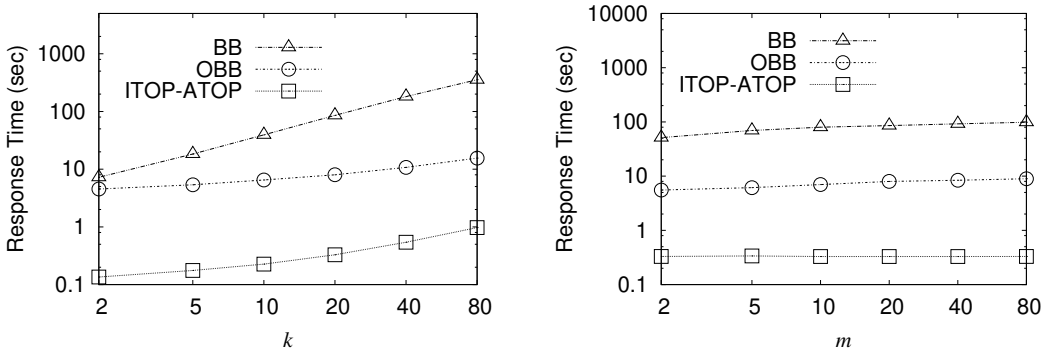
tribution). We can see that our ITOP-ATOP method can consistently beat other methods by 1 to 2 orders of magnitude for varying d. In addition, for different distributions of P for IND F, our method greatly outperforms the BB and OBB, especially in the ANT case where BB and OBB take 3022 and 649 seconds, respectively, while ITOP-ATOP runs in only 0.61 seconds.

In summary, running an all top-k query using our best method ETA is a much better alternative that repetitive executions of RTA if multiple reverse queries are to be evaluated. In addition, evaluating an all top-k query using ETA and using its result to evaluate an $ITOP_k^m$ query is 1-2 orders of magnitude faster than the state-of-the-art method proposed in [22], even if this method is optimized to re-use cached results



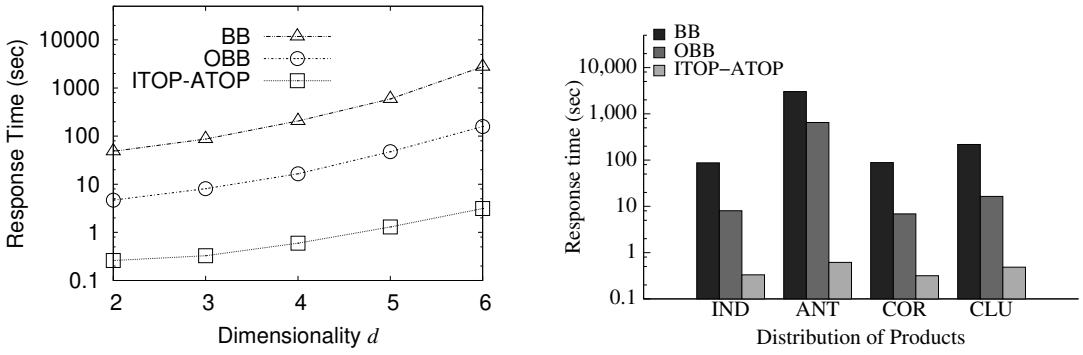
(a) Effect of d (b) Effect of k

Figure 14: Response time of ETA over that of RTA



(a) Effect of k (b) Effect of m

Figure 15: Comparison of different $ITOP_k^m$ approaches



(a) Effect of d (b) Varying distribution of P (IND F)

Figure 16: Comparison of different $ITOP_k^m$ computation

of top-k queries.

7 Related Work

7.1 Top-k queries

Top-k queries [10, 12, 20] provide a convenient way for users to find important objects according to their preferences. In [12], a threshold algorithm (TA) has been proposed to combine object ranks from different sorted lists with a help of an aggregate function f. TA scans the lists sequentially, in a round-robin fashion, and computes the aggregate score of each encountered object, while maintaining the top-k set. As soon as the aggregate scores of the remaining objects cannot exceed the top-k scores found so far, TA terminates. Due to its popularity, many variants of TA have been proposed (e.g., [10]). Onion [9] and PREFER [14, 15] are two top-k methods which rely on pre-processing techniques. Onion [9] pre-computes the convex hulls of the data, and organizes them by layers. Linear top-k computation is then processed by scanning objects incrementally, from exterior layers to interior layers. Onion will stop when it knows that the remaining layers cannot contain any other results. The major disadvantage of Onion is that the costs of pre-computation and query are very expensive, due to the complexity of convex hull computation ($O(n^{d/2})$ in d-dimensional space). Also Onion cannot be used when dataset is frequently updated because re-computations of convex hulls are needed in this case. PREFER [14, 15] first generates materialized views; a top-k query is answered by scanning the views with most similar preferences of the query. [15] proposed an algorithm to determine the best views to be materialized when top-k queries are pipelined. However, as demonstrated in [15], we need to materialized many views before we can ensure satisfactory performance. In addition, PREFER is only suitable for static data, which is the same as Onion. The performance of Onion can be improved with the use of robust indexing [25]. However, building such robust indexing [25] is rather expensive.

BRS [20] is a branch-and-bound approach to answer top-k queries over a set of objects that are indexed by an R*-Tree. BRS uses a heap to maintain candidate entries, traversing the R*-tree in a top-down manner. At every iteration, BRS fetches the best entry from the heap. If the entry is a leaf entry of the R*-tree, then it is output as the next result in the ranking; the algorithm stops if we have enough results. If the entry is in an intermediate node, then the corresponding node is accessed and for each of its entries e a *max score* is computed and e is inserted into the heap. As shown in [20], BRS is an I/O optimal algorithm which means that it accesses only the tree nodes which may contain the top-k results. Since *max score* is a general concept, this algorithm can be applied to both monotone and non-monotone preference functions.

Recently, a *group recommendation* problem has been studied in [5]. Given a group of people, a *consensus relevance score* function is used to model the interests and preferences of all group members. The score of an object is defined as a linear combination of *group relevance* and *group disagreement*. Using the monotonicity of relevance and disagreement, a TA-like algorithm is designed for top-k processing. This paper shares the same intuition with our paper to recommend products to a group of users. However, we focus on providing different recommendations to different users based on their individual preferences, while the goal in [5] is to provide a consensus recommendation of all users. Another technical difference is that our methods are designed for computing multiple top-k queries simultaneously for a large number ($\sim$10K) of users, while the group size in [5] is very small ($<$10). The proposed solution in [5] is obviously inapplicable to our problem.

7.2 Other related queries

As discussed in Section 2, reverse top-k [21] and top-m influence queries [22] have been recently proposed to assess the influence of an object and find the most influential objects, respectively. We showed in this paper that all top-k search can be used to answer these queries efficiently.

There is plenty of work on skyline evaluation (e.g., [8, 19, 27]). The concept of skyline is based on the dominance relationship. The objective is to find the objects that are not dominated by others. The skyline operator was first proposed in [8]. Papadias et al. [19] proposed an incremental skyline algorithm that access a minimal number of nodes from an R*-Tree that indexes the data. An object-based space partitioning method that provides efficient skyline computation in high dimensional spaces was proposed in [27].

Several novel types of queries have been proposed recently to assist the analysis tasks of product manufacturers. [16] was the first paper to use the concept of dominance for business analysis from a microe-

conomic perspective. A data cube model (DADA) is proposed to summarize the dominance relationships between objects in all combinations of dimensions. The space is modeled by the grid (i.e., matrix) of dimensional value combinations (assuming that features have small integer domains) and each cell summarizes the dominance of products in it. In [23], the problem of creating competitive products have been studied. In [24], the authors aim at finding the best sub-space for a query object where it is highly ranked. Miah et al. [17] studied an optimization problem that selects a subset of attributes of a product t such that t's shortened version still maximizes t's visibility to potential customers.

8 Conclusion

In this paper, we studied the problem of batch evaluation of numerous top-k queries (all top-k queries, $ATOP^k$). To our knowledge, this is the first thorough study for this problem. We proposed two batch processing techniques; the first is a *batch indexed nested loops* approach and the second is a *views-based threshold* algorithm with a set of optimization techniques, including *d-bounding views*, *simplex partitioning*, and *batch objects accessing*. We demonstrated that $ATOP^k$ queries can be used to boost the performance of reverse top-k and top-m influential queries. In the future, we plan to study alternative techniques for $ATOP^k$ queries that employ parallel processing and improve the memory management for ETA (e.g., by finding an appropriate order to evaluate the simplexes). Moreover, we intend to study additional queries that can make use of $ATOP^k$ as a module.

References

[1] The Internet Movie Database `http://www.imdb.com/interfaces`.

[2] Retrevo Survey `http://www.retrevo.com/content/blog/2010/11/holiday-shopping-trends-and-black-friday-special-report`.

[3] NBA Basketball Statistics `http://www.databasebasketball.com/`.

[4] Household dataset `http://www.ipums.org/`.

[5] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[6] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD Conference*, pages 322–331. ACM Press, 1990.

[7] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Transactions on Information Theory*, 15(6):658–664, 1969.

[8] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[9] Yuan-Chi Chang, Lawrence D. Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. The Onion Technique: Indexing for Linear Optimization Queries. In *SIGMOD Conference*, pages 391–402, 2000.

[10] Surajit Chaudhuri and Luis Gravano. Evaluating Top-*k* Selection Queries. In *VLDB*, pages 397–410, 1999.

[11] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. Answering Top-k Queries Using Views. In *VLDB*, pages 451–462, 2006.

[12] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.

[13] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[14] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *SIGMOD Conference*, pages 259–270, 2001.

[15] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.*, 13(1):49–70, 2004.

[16] Cuiping Li, Beng Chin Ooi, Anthony K. H. Tung, and Shan Wang. DADA: a data cube for dominant relationship analysis. In *SIGMOD Conference*, pages 659–670, 2006.

[17] Muhammed Miah, Gautam Das, Vagelis Hristidis, and Heikki Mannila. Standing Out in a Crowd: Selecting Attributes for Maximum Visibility. In *ICDE*, pages 356–365, 2008.

[18] James Munkres. *Elements of Algebraic Topology*, chapter 1.1. Prentice Hall, 2 edition, January 1984.

[19] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[20] Yufei Tao, Dimitris Papadias, Vagelis Hristidis, and Yannis Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32:424–445, 2007.

[21] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse Top-*k* Queries. In *ICDE*, pages 365–376, 2010.

[22] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Identifying the Most Influential Data Objects with Reverse Top-*k* Queries. *PVLDB*, 3(1):364–372, 2010.

[23] Qian Wan, Raymond Chi-Wing Wong, Ihab F. Ilyas, M. Tamer Özsu, and Yu Peng. Creating Competitive Products. *PVLDB*, 2(1):898–909, 2009.

[24] Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han. Promotion Analysis in Multi-Dimensional Space. *PVLDB*, 2(1):109–120, 2009.

[25] Dong Xin, Chen Chen, and Jiawei Han. Towards Robust Indexing for Ranked Queries. In *VLDB*, pages 235–246, 2006.

[26] Jun Zhang, Nikos Mamoulis, Dimitris Papadias, and Yufei Tao. All-nearest-neighbors queries in spatial databases. In *SSDBM*, pages 297–306, 2004.

[27] Shiming Zhang, Nikos Mamoulis, and David W. Cheung. Scalable skyline computation using object-based space partitioning. In *SIGMOD Conference*, pages 483–494, 2009.

APPENDIX

A Proof of Theorems

Theorem 1.

Proof. For any unseen object q, we know that $v_i(q) \leq v_i(\phi)$ since we have accessed all objects p in v_i that $v_i(p) > v_i(\phi)$. According to Definition 6, each function $f \in F$ can be represented by $\sum_{i=1}^{d} r_i v_i$ where

$r_1, \ldots, r_d > 0$. Therefore,

$$\begin{aligned} f(q) = f \cdot q &= (r_1 v_1 + \ldots + r_d v_d) \cdot q \\ &= r_1(v_1 \cdot q) + \ldots + r_d(v_d \cdot q) \\ &\leq r_1 v_1(\phi) + \ldots + r_d v_d(\phi) \\ &\quad \text{Since } r_1, \ldots r_d \geq 0 \\ &= r_1(v_1 \cdot \phi) + \ldots + r_d(v_d \cdot \phi) \\ &= (r_1 v_i + \ldots + r_d v_d) \cdot \phi \\ &= f(\phi) \end{aligned}$$

□

Theorem 2.

Proof. Given a function f, a hyperplane $\mathcal{HP}$ and the maximum point $o = (1, \ldots, 1)$, the cross point f^t of f on the $\mathcal{HP}$ is a $(d-1)$ dimensional point. This cross point can be expressed by $f^t = o + \alpha f$ and α is a scalar that can be obtained using the equation of the hyperplane. Without loss of generality, we choose d dimensional hyperplane $\mathcal{HP}$ to be the one with all points p having its d-th value $p[d] = 0$ which means $\mathcal{HP}$ is orthogonal to the d-th dimension. It also implies that $p \cdot h = p \cdot (0, \ldots, 0, 1)^T = 0$. We consider the non-degenerate case that $o \notin \mathcal{HP}$.

Suppose we have a set of d dimensional views vectors $\{v_i\}$, this mapping to $(d-1)$ space can be done in two steps. First we find the cross point $v_i^t = o + \alpha_i v_i$ ($\alpha_i \neq 0$) and $v_i^t \cdot h = 0$. Then we transform v_i^t by $\mathbf{M}_{(d-1)\times d} = (\mathbf{I}_{d-1}\ 0)$, and get d-1 point $v_i^t = \mathbf{M}v_i^t = \alpha_i \mathbf{M}v_i + \mathbf{M}o$. Notice that if we set $\mathbf{M}^t_{d\times(d-1)} = \begin{pmatrix} \mathbf{I}_{d-1} \\ 0 \end{pmatrix}$, then we can get $v_i = \mathbf{M}^t v_i^t$ back. So we can get $v_i = (1/\alpha_i)(v_i^t - o) = (1/\alpha_i)\mathbf{M}^t v_i^t - (1/\alpha_i)o$.

Assume that we have a set of views $\{v_i\}$ and a preference function f where their mapping points on the $\mathcal{HP}$ are denoted as $\{v_i^t\}$ and f^t. Also we assume that their corresponding α is not equal to 0 and have the same sign (i.e., $\forall \alpha_i > 0$).

IF side. If $f^t \in \Delta^{d-1}$, then $f^t = \sum_{i=1}^d r_i^t v_i^t$, where $\sum_{i=1}^d r_i^t = 1$ and $r_i^t \geq 0$. Therefore,

$$\begin{aligned} f &= (1/\alpha_f)\mathbf{M}^t f^t - (1/\alpha_f)o \\ &= \sum_{i=1}^d r_i^t (1/\alpha_f)\mathbf{M}^t v_i^t - (\sum_{i=1}^d r_i^t)(1/\alpha_f)o \\ &= \sum_{i=1}^d r_i^t ((1/\alpha_f)\mathbf{M}^t v_i^t - (1/\alpha_f)o) \\ &= \sum_{i=1}^d (r_i^t \alpha_i/\alpha_f) v_i \end{aligned}$$

So by choosing $r_i = r_i^t \alpha_i/\alpha_f$, then we can have $f = \sum_{i=1}^d r_i v_i$, and all $r_i \geq 0$. Thus f is bounded by $\{v_i\}$.

ONLY IF side. If $f = \sum_{i=1}^d r_i v_i$, and all $r_i \geq 0$, then we have $f \cdot h = \sum_{i=1}^d r_i v_i \cdot h$. Therefore,

$(f^t - o) \cdot h = \sum_{i=1}^{d}((r_i\alpha_f)/\alpha_i)(v_i^t - o) \cdot h$, which means $\sum_{i=1}^{d} r_i\alpha_f/\alpha_i = 1$. On the other hand,

$$\begin{aligned} f^t &= \alpha_f \mathbf{M} f + \mathbf{M} o \\ &= \sum_{i=1}^{d} r_i \alpha_f \mathbf{M} v_i + \mathbf{M} o \\ &= \sum_{i=1}^{d} r_i \alpha_f / \alpha_i (\mathbf{M}\mathbf{M}^t v_i^t - \mathbf{M} o) + \mathbf{M} o \\ &= \sum_{i=1}^{d} r_i \alpha_f / \alpha_i v_i^t \end{aligned}$$

Then by choosing $r_i^t = r_i\alpha_f/\alpha_i$, we know that $f^t = \sum_{i=1}^{d} r_i^t v_i^t$, while $\sum_{i=1}^{d} r_i^t = 1$ and $r_i^t \geq 0$. Thus $f^t \in \Delta^{d-1}$. □