Efficient All Top-*k* Computation

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

Shen Ge, Leong Hou U, Nikos Mamoulis, David W. Cheung

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 algorithm. We propose appropriate optimization techniques for the two approaches and demonstrate experimentally that the second approach is consistently the best. Our approach facilitates evaluation of other complex queries that depend on the computation of multiple top-*k* queries, such as reverse top-*k* and top-*m* influential queries. We show that our batch processing technique for these complex queries outperform the state-of-the-art by orders of magnitude.

Index Terms—all top-k queries, view-based index

1 Introduction

Many real life applications support ranking of products according to user preference functions. For example, consider an online store (e.g., Amazon), which ranks blu-ray discs according to the preferences of customers. Preferences could be explicitly expressed by each user, or implicitly derived from user purchase records. Preferences are typically defined on some product features. For example, blu-ray discs could be ranked based on their *movie cast* and *release date*; recent movies having a good cast rank higher than others. To simplify illustration and analysis, in the rest of the paper, we assume that product 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, p_3, and p_4 are used to represent four products respectively in Fig. 1. Modeling objects in such a multidimensional space is common for diverse types of queries, such as top-k queries [1], [2], [3], skyline queries [4], [5], and market analysis queries [6], [7].

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 $(0, 0)$. Different customers may have completely different preferences. For instance, f_b represents the preferences of a customer, u_b, who is concerned more about the quality of casting than the release date. Accordingly, p_2 is the best product according to u_b's preferences. Without loss of generality, we assume that all preference functions f are linear and and the coefficients of them 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.

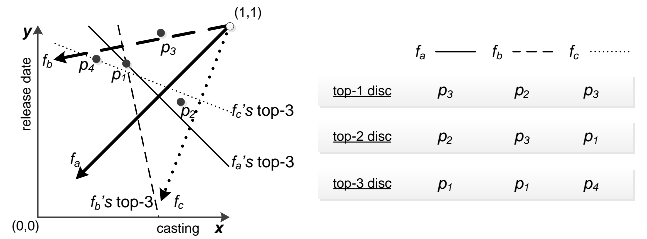


Fig. 1. Top-k queries for online stores

Generally speaking, users are more interested in top ranked products. Given a constant k, in addition with a ranking function f, a top-k query [1], [2], [3] returns the k highest ranked objects according to f. For example, consider the four products in Fig. 1. For $k = 3$ and user u_a, whose preferences are captured by the linear function $f_a = 0.5x + 0.5y$, the result of the top-k query is $TOP^3(f_a) = \{p_3, p_2, p_1\}$.

Many applications have millions of users and nu-

- *S. Ge, N. Mamoulis and D. W. Cheung are with the Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong. E-mail:* {sge,nikos,dcheung}@cs.hku.hk
- *L. H. U is with the Department of Computer and Information Science, University of Macau, Av. Padre Tomás Pereira Taipa, Macau. E-mail:* ryanlhu@umac.mo

0000–0000/00$00.00 © 20XX IEEE

merous top-k queries may have to be evaluated simultaneously. Recommendation systems of online stores is such an application (i.e., recommendations to numerous users currently online). As another example, consider a second-hand cars company, which recommends 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 result 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. We call this problem the *all top-k query*, $ATOP^k$.

To the best of our knowledge, there is no efficient approach to compute multiple top-k queries simultaneously. 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 *view-based threshold* algorithm. We also propose several novel optimization techniques for these methods.

Besides products recommendation, other tasks, such as product promotion analysis [8] and identifying the most influential products [9], can benefit from an efficient approach for computing multiple top-k queries simultaneously, as we discuss in Section 3. We demonstrate the utility of our result in these complex analysis tasks; when $ATOP^k$ is used as a search module for reverse top-k [8] and top-m influential [9] queries, the evaluation cost of these queries greatly decrease.

The rest of the paper is organized as follows. We provide formal definitions and review preliminary concepts in Section 2. The applicability of $ATOP^k$ in the evaluation of related queries is discussed in Section 3. An intuitive batch processing technique is introduced in Section 4. In Section 5, we present an alternative batch processing approach which extends the view-based threshold algorithm [10] and fully optimize it. Section 6 discusses how we can use our techniques to support related queries, including reverse top-k and top-m influential queries. In Section 7, we experimentally evaluate our methods using synthetic and real data. Section 8 discusses related work. Finally, Section 9 concludes the paper.

2 PRELIMINARIES

This section includes all formal definitions and preliminary concepts, based on which we build our solutions. We begin by defining top-k and all top-k queries.

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 $f(p_i) \geq f(p_j)$, $\forall p_i \in TOP^k(f), p_j \in P \backslash TOP^k(f)$.

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 $f \in F$.

The *reverse top-k query* [8] is a derived concept. Given a product p_i and a set of user preferences, a reverse top-k query, $RTOP^k(p_i)$, returns the users who have p_i in their top-k results (Definition 3). 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. Product promotion is an application of $RTOP^k(p_i)$. Assume 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 would highly rank p_i, based on their known preferences.

Definition 3 (Reverse top-k query): 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 $f(p) \geq f(q)$.

The problem of finding the most influential products has been recently studied by Vlachou et al. [9]. The *influence score* $I^k(p_i)$ of a product p_i (Definition 4) is defined by the number of customers who have p_i in their top-k preferences.

Definition 4 (Influence score, I^k): 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 influential query [9], $ITOP_k^m$, finds the m most influential products (Definition 5). Ranking is based on the influence scores I^k. $ITOP_k^m$ finds 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.) can help companies to assess the popularity of their current products and/or design new ones with features similar to the most popular products. For instance, the iPad is considered a good product because it is ranked highly by many customers in a survey [11]. 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).

Definition 5 (Top-m influential query): 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 this paper, we study $ATOP^k$ and show how it can be used as a module for efficient evaluation techniques for $RTOP^k(p)$ and $ITOP_k^m$. Table 1 summarizes the notation used throughout the paper. Our solution builds on methods for top-k queries using materialized ranking views [10]. A materialized ranking view is simply the result of a top-k query. Das et al. [10] proposed a Linear Programming adaptation of the Threshold Algorithm (LPTA), which extends the Threshold Algorithm (TA) [2] to apply on views. LPTA sequentially accesses the results of two or more materialized object rankings, based on different views, in order to compute the top-k objects of 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 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 [10]. We illustrate this process by an example in Section 5.

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

3 APPLICATIONS OF $ATOP^k$ AS A MODULE

Besides its direct applications (e.g., in recommender systems), discussed in the Introduction, $ATOP^k$ can also be used as a processing module of other queries. We note that the solution for reverse top-k problem proposed in [8] 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. Also in [9], 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 [8], 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 [9], 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 [8] and [9], 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.

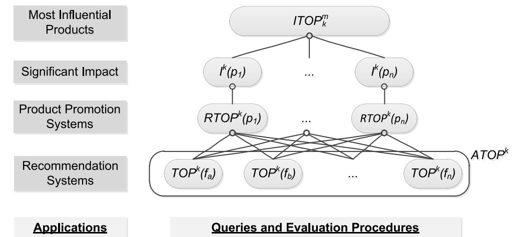


Fig. 2. Relationship of different queries

4 BATCH TOP-k PROCESSING

Top-k queries are extensively studied in the literature [1], [2], [3], [10]. 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.

This method can be considered as the counterpart of block indexed nested-loops in relational databases and spatial join queries in spatial databases [12]. Suppose that the objects are indexed by a multidimensional index, e.g., R*-tree [13], 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 [14] 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 7, we study the choice of δ and evaluate alternative grouping strategies.

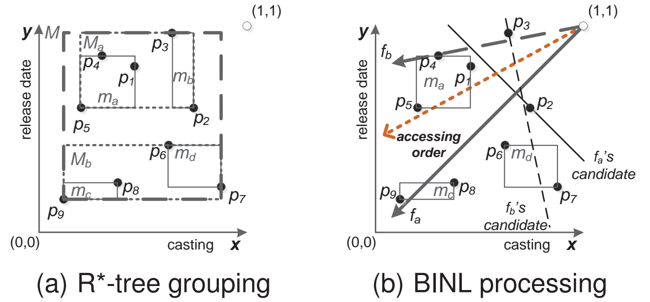
Let F_g be a group of functions; the group maximum score $s^{F_g}_{max}(p)$ of an object p computed by the functions of the group is $s^{F_g}_{max}(p) = \sum_{i=1}^{d} \max_{f_g \in F_g}\{f_g[i]\}p[i]$. For a given F_g, we traverse the nodes and objects in the R*-tree (e.g. Fig. 3(a)) in descending order of the group maximum score. We first load the root of the R*-tree, calculating $s^{F_g}_{max}$ of all entries in it (i.e., for each minimum bounding rectangle (MBR)). The maximum possible score $s^{F_g}_{max}(m)$ 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 with the largest values in all dimensions 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 against all the points in that leaf node and the candidate lists of those functions in F_g are updated accordingly. 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, ..., F_g} by Hilbert curve
2:  for all F_i ∈ {F_1, ..., 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 ∈ m do
8:                  compute the maximum possible score s^{F_i}_max(m_i) to m_i
9:                  en-heap (m_i, s^{F_i}_max(m_i)) into PQ
10:         else if m is a leaf MBR then
11:             for all f_i ∈ 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
14:                     update the candidate list of f_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 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 $s^{F_g}_{max}$ 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; p_2 and p_3 have already been seen by f_a and f_b and we 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 $f_a(m_d) < f_a(p_2) < f_a(p_3)$.



(a) R*-tree grouping (b) BINL processing

Fig. 3. An example of batch indexed nested loops

Discussion. Techniques similar to BINL have been proposed before for All Nearest Neighbors Queries (ANN) in spatial databases [12]. 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.

5 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 [15]. As discussed in Section 2, LPTA [10] can be used to compute top-k queries using views. Here 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. Assuming that v_1 and v_2 have been accessed 2 times respectively, the regions being accessed are shaded in the figure. Note that the unseen region must be convex if all view 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 two sorted accesses from each view, 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 $s_{max}(f_a)$ (computed by linear programming) 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

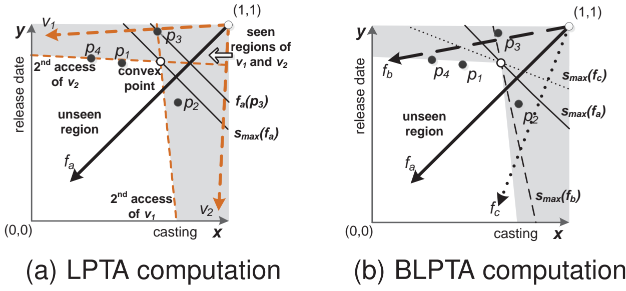


Fig. 4. Top-k computation using ranking views

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.

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

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 2 accesses from each of the views. The *maximum possible scores*, s_{max}, of the functions are shown by three different lines in Fig. 4(b). In this example, all functions are marked as *stopped* after the 2nd access from each view since the corresponding s_{max} score is no better than the candidate score. Therefore, BLPTA exits the while-loop and returns the all top-k result.

BLPTA terminates early if all functions are marked as *stopped*. However, this method is costly 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 the remainder of this section, we discuss and resolve these three issues and propose an optimized version of the algorithm.

5.1 Avoiding linear programming

Given a set of pre-computed views V, BLPTA (and LPTA as well) can compute the top-k queries using a subset of V and the selection can be determined by the cost estimation technique suggested in [10]. However, the maximum possible score is still computed by linear programming. Considering the fact that this computation will be carried out for all *running* preference functions against all *accessed* objects, it easily becomes the bottleneck. Motivated by this, we first redesign our method to avoid linear programming computation. Instead of using a subset of precomputed views, we *construct the views* based on some constraints, such that the maximum possible score can be derived from the cross point of d *hyperplanes* (technique to be discussed shortly). We now 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. 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.

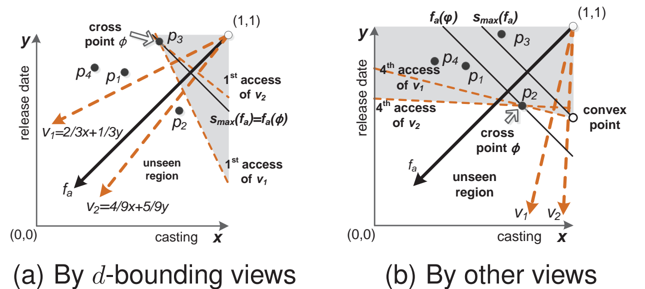


Fig. 5. Example of different views settings

Besides, we define as the *scanning hyperplane* of a view v, the hyperplane which is perpendicular to 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 $v[1]x[1] + \ldots + v[d]x[d] = s$.

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 (i.e., $s_{max}(f_a) < f_a(\phi)$).

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, $f(\phi)$ can be viewed as 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. For every view v_i and its last seen score s_i, we have

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

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:

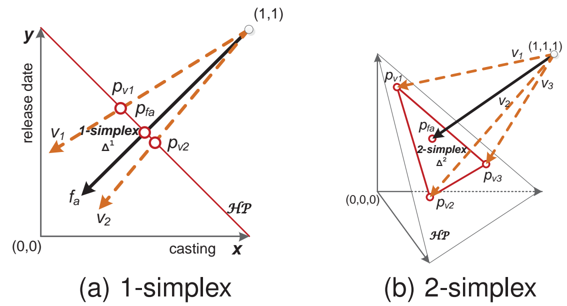
$$\phi = \begin{pmatrix} v_1[1] & \ldots & v_1[d] \\ \vdots & \vdots & \vdots \\ v_d[1] & \ldots & v_d[d] \end{pmatrix}^{-1} \begin{pmatrix} s_1 \\ \vdots \\ s_d \end{pmatrix}$$

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 three objects in total from the views in Fig. 4(a). However, finding the tightest d-bounding views is equivalent to a problem of finding minimum volume enclosing simplices [16], which is NP-hard. 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 the next section, we study how to tighten these views by a partitioning technique.

5.2 View-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 intersecting 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 ($\Delta^{d\text{-}1}$) [17] 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 simplices 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 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 based on the corresponding points from v_1, v_2, and v_3. We can partition the Δ^2 into three sub-simplices (i.e., Δ^2_1, Δ^2_2, and Δ^2_3) by adding view v_4 (see Fig. 7(b)).



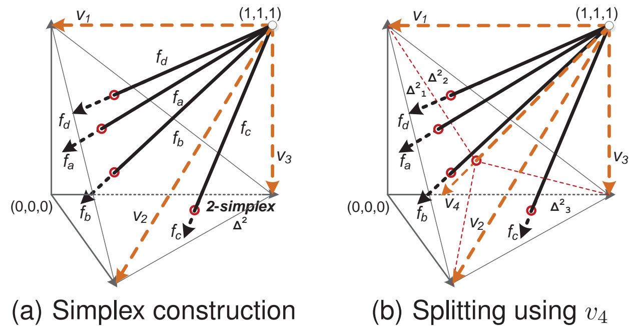
(a) 1-simplex (b) 2-simplex

Fig. 6. Examples of d-bounding views projection

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 $\{v_2,v_3,v_4\}$, which 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.



(a) Simplex construction (b) Splitting using v_4

Fig. 7. An example of partitioning

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 5.1. 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 number of functions being bounded by it is less than a ratio λ of the total.

The partitioning procedure is described by 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 $\Delta^{d\text{-}1}.F$ ($\Delta^{d\text{-}1}.F$ denotes the associated preference function set F of the simplex Δ^{d-1}). Lines 3–12 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 partition the simplex $\Delta^{d\text{-}1}$ into d sub-simplices using Definition 7 (line 5). Every bounding function f of $\Delta^{d\text{-}1}$ is assigned to one of the d sub-simplices. 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 (controlled by parameter λ).

Algorithm 3 d-bounding views partitioning

```
    Algorithm partitioning(V, F, λ)
 1: construct Δ^{d-1} for V and set Δ^{d-1}.F := F
 2: push Δ^{d-1} into a queue Q
 3: while Q is not empty do
 4:     Δ^{d-1} := Q.pop()
 5:     partition Δ^{d-1} into {Δ_1^{d-1}, ..., Δ_d^{d-1}} using v_avg := AVG v_i ∈ V
 6:     for all f ∈ Δ^{d-1}.F do
 7:         assign f to Δ_i^{d-1} if f is in the interior of Δ_i^{d-1}
 8:     for all Δ_i^{d-1} ∈ {Δ_1^{d-1}, ..., Δ_d^{d-1}} do
 9:         if size(Δ_i^{d-1}.F) ≥ λ · size(F) then
10:             push Δ_i^{d-1} into Q                    ▷ further partition Δ_i^{d-1}
11:         else
12:             Δ_G := Δ_G ∪ {Δ_i^{d-1}}
13: return Δ_G
```

5.3 Simplex execution order

Even through the simplices (generated by Algorithm 3) can be evaluated independently at any order, the memory usage can be controlled better if the execution order is well designed. According to our partitioning approach, each simplex contains d views and each view is used by multiple simplices. A view can be removed from memory after all relevant simplices have been evaluated. To minimize the total memory usage, we should define an execution order such that the maximum number of views kept in memory is minimized. Finding the optimal order is a combinatorial problem, therefore we adopt a greedy approach, where the next simplex is decided by the views kept in memory. Intuitively, a view v_{min} should be cleaned up first if v_{min} is used by the fewest simplices among all views in memory. In other words, we first evaluate all simplices that use v_{min}, in order to remove v_{min} from memory as early as possible. The effectiveness of this approach is demonstrated in our experiments.

5.4 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 4). However, the objects being accessed from views are unavoidably evaluated by the functions in BLPTA. For the sake of batch pruning, we fetch a set of objects from a view instead of one object at each access. In order to have stable performance at different data distributions, we stop fetching objects from a view if the volume of the accessed objects' MBR is larger than a threshold ω. In addition, we apply the same pruning idea as BINL, i.e., not every object is necessarily evaluated by the functions, improving pruning effectiveness.

5.5 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; each of group is bounded by a corresponding set of d-bounding views (see Section 5.2). Given the execution order of the groups (see Section 5.3), we evaluate the functions in batch using the corresponding d-bounding views. At every iteration, for each group, we access the views in a round-robin fashion. At each access, we fetch multiple objects from the views, until the MBR m of them has a larger volume than ω (see Section 5.4). Subsequently, we update the cross point ϕ of d scanning hyperplanes (see Section 5.1).

In the next step, we examine whether the objects belonging to m should be examined by a function using the MBR pruning technique (see Lemma 1 in Section 4). 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 5.1). The all top-k results of a group are found as soon as all functions in the group are marked as *stopped*. Algorithm 4 is a detailed pseudocode for ETA.

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 using an off-the-shelf top-k computation algorithm (e.g., BRS [3]). In order to reduce memory consumption of computed view rank lists, the memory held for a view is released after the view is no longer needed.

5.5.1 Cost Simulation Analysis

We observe that the benefit of tightening the views (i.e., minimizing λ) drops proportionally to the size of simplices in ETA. To demonstrate this, we propose

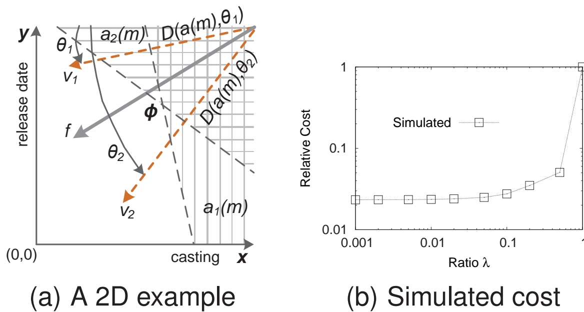
Algorithm 4 ETA Algorithm

Algorithm $ETA(V, P, F, k, \omega, \lambda)$
1: **for all** $f \in F$ **do**
2: $TOP^k(f) \leftarrow \emptyset$ and mark f as *running*
3: $\Delta_G := partitioning(V, F, \lambda)$ ▷ Section 5.2
4: **while** Δ_G is not empty **do**
5: $\Delta_i^{d\text{-}1} := select(\Delta_G)$; $\Delta_G := \Delta_G - \{\Delta_i^{d\text{-}1}\}$ ▷ Section 5.3
6: **while** $\Delta_i^{d\text{-}1}.F$ is not empty **do**
7: **for all** $v \in \Delta^{d\text{-}1}.V$ **do**
8: fetch objects until their MBR volume $V_m \geq \omega$ ▷ Section 5.4
9: compute cross point ϕ using d-scan hyperplanes ▷ Section 5.1
10: **for all** $f \in \Delta_i^{d\text{-}1}.F$ **do**
11: **if** $f(m)$ is better than k-th score in $TOP^k(f)$ **then**
12: **for all** $p \in m$ **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** $f(\phi)$ is no better than k-th score in $TOP^k(f)$ **then**
16: mark f as *stopped* and remove f from $\Delta_i^{d\text{-}1}.F$

a model that simulates the accessing cost for different view settings (i.e., represented by their angles) using 2D data. In Figure 8(a), we illustrate two different views (v_1 and v_2) where their angles to the top horizontal line are θ_1 and θ_2, respectively. For the sake of analysis, we assume that the objects are uniformly distributed in the domain area. Based on this assumption, the scanned area $a(m)$ of accessing a specific number of objects (i.e., m objects) is the same for any view/function (i.e., $a_1(m) = a_2(m)$).

Given the scanned area $a(m)$, the cross point ϕ of the scanning hyperplanes (computed by $a(m)$), and a bounded user preference function θ, we can calculate the minimum accessed distance $D_{min}(a(m), \theta, \phi)$ of the user preference in the unseen region by LPTA. Besides, given the angle θ and the scanned area k/N of a view/function, we can compute the accessed distance, $D(k/N, \theta)$, by simple geometry.

To determine the cost for a specific user function f (i.e., represented by θ), we need to count the number of accessed objects m from each view such that the top-k score is not worse than maximum possible score (i.e., minimum accessed distance). This can be modeled by $D_{min}(a(m), \theta, \phi) \leq D(k/N, \theta)$, where m can be calculated given the values of k, θ, and ϕ.



(a) A 2D example (b) Simulated cost

Fig. 8. Cost analysis

Figure 8(b) shows our simulation result as a function of λ where the value of θ and ϕ can be derived by λ and k is set to 20. The relative cost decreases as λ decreases; however, when λ is smaller than 0.02, the benefit of further partitioning of the simplices is not significant. On the other hand, we have to compute more views if we decompose more simplices. It is clear that we should stop our simplex partitioning at some point by considering the tradeoff between these two factors. In this work, we set λ to 0.02 based on both numerical and experimental analyses (see Section 7).

6 Efficient Reverse Top-k and Top-m Influential Computation

In this section, we show how we can use our $ATOP^k$ algorithms to facilitate the evaluation of reverse top-k $RTOP^k(p)$ and top-m influential $ITOP_k^m$ queries. First, we briefly review the state-of-the-art solutions to these problems from [8] and [9].

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

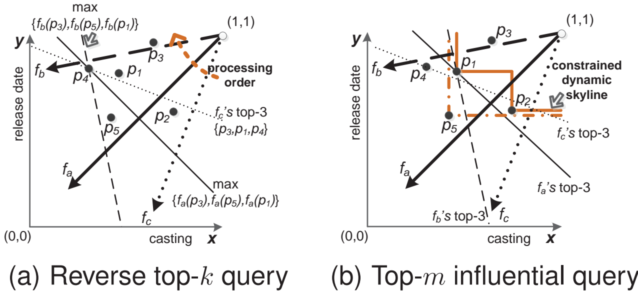
Given a set of objects P and a set of preference functions F, the reverse top-k query of an object $p \in P$ returns the subset of F that contains p in their top-k result. A naïve method computes a reverse top-k query by evaluating the preference functions one by one. [8] proposed evaluating the functions 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 [8]. However, this process might evaluate all functions, in the worst case.

We demonstrate the reverse top-k computation in Fig. 9(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.

6.2 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

1. Closeness can be measured by a cosine function.



(a) Reverse top-k query (b) Top-m influential query

Fig. 9. Examples of other queries

top-k queries. The cost becomes too high if F and P are large. In [9], 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. 9(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., [3]), the first m de-heaped objects are the result of the query. This BB algorithm is the best approach in [9] and it is much faster than the straightforward solution. However, BB 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.

6.3 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. 9, the $ATOP^k$ results are shown in Fig. 10(a). By "inverting" this table, as shown in Fig. 10(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

Fig. 10. 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 [8] and [9], many top-k queries are evaluated if F and P are large, while in [9] 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 7, OBB is much slower than our "$ITOP_k^m$ using $ATOP^k$" approach.

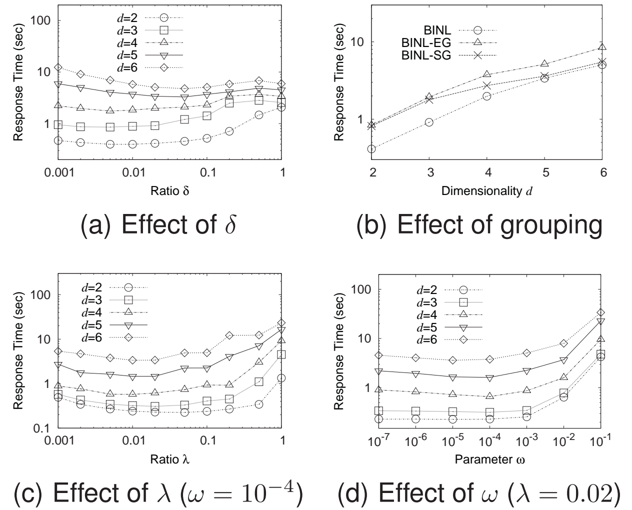
7 EXPERIMENTAL EVALUATION

According to the methodology in [4], 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 [18] and Household [19]. 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 Ubuntu 11.04. Table 2 shows the ranges of the investigated parameters. In each experiment, we vary a single parameter, while setting the others to their default values (shown in bold in Table 2). Our system uses a 4KB page size. In order to measure the exact I/O cost, we assume no memory buffer is available.



(a) Effect of δ (b) Effect of grouping

(c) Effect of λ ($\omega = 10^{-4}$) (d) Effect of ω ($\lambda = 0.02$)

Fig. 11. Sensitivity experiments

Parameter sensitivity experiments. We first study the effect of various tuning factors on the algorithms, BINL and ETA. We investigate the effect of δ (grouping ratio in BINL), the effect of different grouping strategies for BINL, λ (splitting ratio in ETA), and ω (size of accessed objects' MBR in ETA).

Figure 11(a) shows the effect of δ on the cost of BINL for different dimensionality values d. For very small δ values, the cost is high since forming either 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. Regarding the function grouping strategy in BINL, we compare Hilbert curve ordering to cosine similarity based grouping (BINL-SG) (proposed in [9]) and ETA bounding view partitioning (BINL-EG) (proposed in this paper), in Figure 11(b). The result shows that Hilbert grouping (BINL) outperforms the other two methods for varying dimensionality d, justifying grouping the function vectors by Hilbert curve ordering in our implementation.

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 $\omega = 10^{-4}$ 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 4) and ETA (Section 5.5) compared to the naïve approach and a simple skyline based solution (Skyband). The naïve approach evaluates the top-k queries one-by-one using BRS [3]. Skyband first collects the objects in the k-skyband (using BBS [5]) and then evaluates the top-k queries one-by-one over it.

Fig. 12(a) shows the response times of the four 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, 2.5 and 1.5 times faster than Naïve, Skyband, and BINL, respectively in all experiments. The skyline-based approach does not scale well with dimensionality due to the increasing number of objects in the skyband. For large values of d, the gap between BINL and ETA becomes smaller, because the MBRs that group multiple accessed objects in ETA becomes too large, reducing the effect of the MBR pruning technique.

Fig. 12(b) compares performance as a function of k. ETA is at least 8.6, 3.48, 2.1 times faster than Naïve, Skyband, and BINL, respectively. All methods are sensitive to k since the problem becomes harder as k increases.

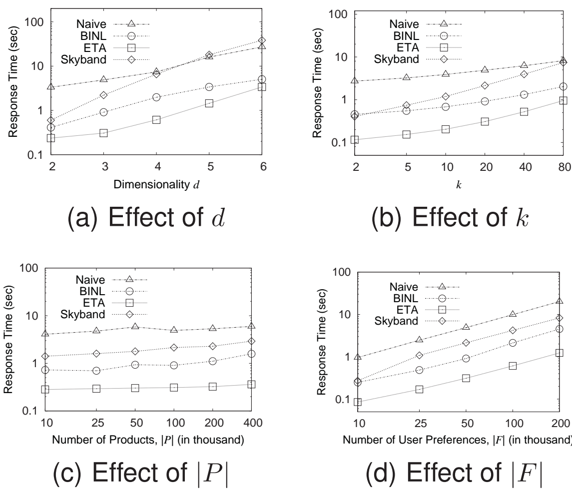
The response times for different numbers of products $|P|$ are shown in Fig. 12(c). The cost is not very sensitive to $|P|$ since the products are indexed and we only need to access a small fraction of the data.

Fig. 12(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. Our ETA still performs the best, followed by BINL, Skyband, and Naïve.

Data distribution. As shown in Fig. 13(a), ETA is at least one order of magnitude faster than Naïve and 2.6 times faster than BINL for different data distributions of P and independently distributed F. ANT distributed objects are the hardest case since top-k computation becomes hard 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. 13(b), ETA is again the best method which is at

TABLE 2
Range of parameter values

Parameter	Values
$\lvert P\rvert$ (in thousand)	10, 25, 50, **100**, 200, 400
$\lvert F\rvert$ (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 volume of accessed objects' MBR, ω	10^{-7}, 10^{-6}, 10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}, 10^{-2}, 10^{-1}

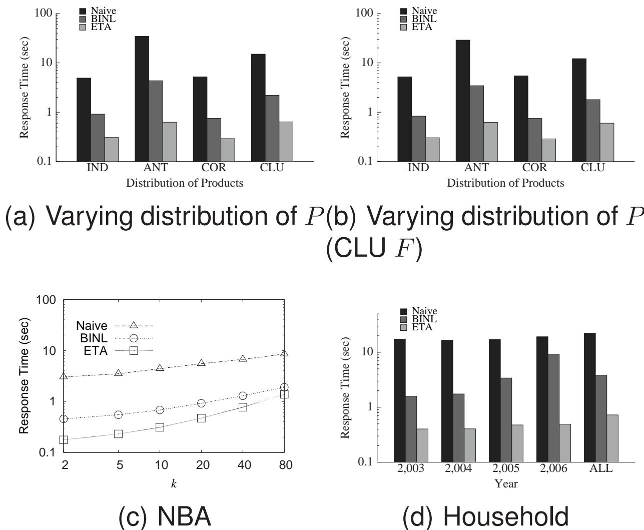


(a) Effect of d (b) Effect of k

(c) Effect of $|P|$ (d) Effect of $|F|$

Fig. 12. Effect of different parameters for $ATOP^k$

least one order of magnitude faster than Naïve and 2.6 times faster than BINL. We conclude that ETA is the best method for all distribution combinations.



(a) Varying distribution of P (b) Varying distribution of P (CLU F)

(c) NBA (d) Household

Fig. 13. Varying data distribution

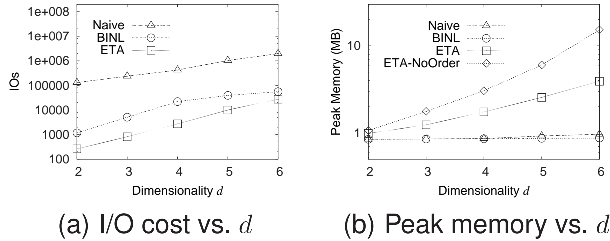
Fig. 13(c) plots the response time of all methods on the NBA real dataset. We instantiated P from this dataset (12,278 records) and set other parameters to their default values. Again, ETA is consistently better than Naïve and BINL for all values of k. Summing up, ETA is the best solution for $ATOP^k$ queries, typically being one order of magnitude faster than Naïve solution and 2-3 times faster than BINL.

In Fig. 13(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, while there are 1.55M different ones in total. We demonstrate the response time of all three methods of the data in these four years as well as the whole data in Figures 13(d). ETA again performs best in all cases, being at least 36 and 4 times faster than Naïve and BINL, respectively. Even though the cardinality of the entire dataset (ALL) is several times larger than that of the yearly datasets, the response time does not increase much, being consistent with the trends in Fig. 12(c).

I/O cost and peak memory usage. Fig. 14(a) and 14(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. 12(a)); ETA accesses several times to two order of magnitude

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

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. Also, if the execution order is randomly selected (ETA-NoOrder), then the execution consumes 3.89 times more memory than ETA at $d = 6$.

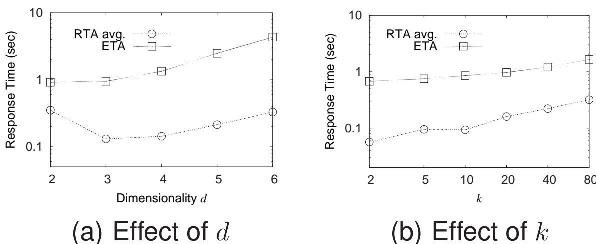


(a) I/O cost vs. d (b) Peak memory vs. d

Fig. 14. Extra experiments for $ATOP^k$

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 [8], [9] to the $ATOP^k$-based alternatives that we introduced in Section 6.3.

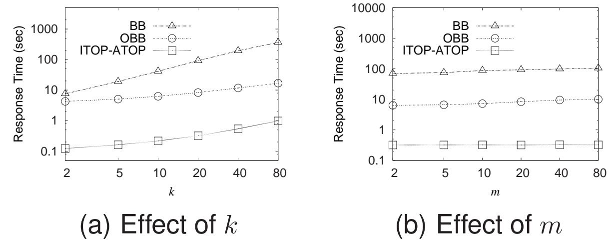
For reverse top-k queries, we plot the response time of $ATOP^k$-based reverse top-k search using ETA vs. the average response time of $RTOP^k$ processing using RTA [8]. The queries are selected randomly from k-skyband objects in order to avoid meaningless results (i.e., no user function considers the query in their top-k result). As shown in Fig. 15(a), RTA is only 2.6 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. 10(b)). In other words, if we are to execute more than 13 $RTOP^k$ queries in $d = 6$, ETA should be preferred to RTA, because the total response time will be better in this case. 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. 15(b)) leads to similar conclusions.



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

Fig. 15. Response time of ETA over that of RTA

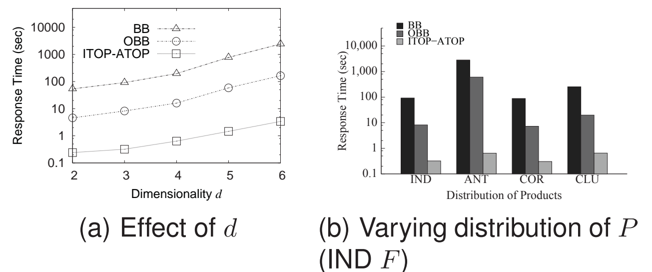
For top-m influential queries, we compare our $ITOP_k^m$ using $ATOP^k$ (ITOP-ATOP) approach (see Section 6.3) to the state-of-the-art solution BB and its optimized version OBB (as discussed in Section 6.3). Fig. 16(a) shows the response time for these methods as a function of k. As k increases, OBB becomes much better than original BB since OBB caches the results of previous top-k computations. However, OBB is still 17 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. 16(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.



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

Fig. 16. Comparison of different $ITOP_k^m$ approaches

Fig. 17 shows some additional experiments on $ITOP_m^k$ queries (varying dimensionality and data distribution). Our ITOP-ATOP method consistently beat other methods by 1 to 2 orders of magnitude for various values of d. In addition, for different distributions of P for IND F, our method greatly outperforms BB and OBB, especially in the ANT case where BB and OBB take 2827 and 607 seconds, respectively, while ITOP-ATOP runs in only 0.64 seconds.



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

Fig. 17. Varying dimensionality and data distribution

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 [9], even if this method is optimized to re-use cached results of top-k queries.

8 Related Work

8.1 Top-k queries

Top-k queries [1], [2], [3] provide a convenient way for users to find important objects according to their preferences. In [2], 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 current top-k set, until it is guaranteed to be the result. Due to its popularity, many variants of TA have been proposed (e.g., [1]). Onion [20] and PREFER [21], [22] top-k methods rely on pre-processing. Onion [20] pre-computes convex hull layers of the data and evaluates linear top-k queries by scanning objects incrementally, from exterior layers to interior layers. Onion stops when it is guaranteed that the remaining layers cannot contain any other results. The high complexity of convex hull computations ($O(n^{d/2})$ in d-dimensional space) makes Onion too expensive to be used in practice. Also Onion cannot be used when dataset is frequently updated because re-computations of convex hulls are needed in this case. PREFER [21], [22] first generates materialized views; a top-k query is answered by scanning the views with most similar preferences to the query. An algorithm to determine the best views to be materialized when top-k queries are pipelined is proposed. However, as demonstrated in [22], we need to materialized many views before we can ensure satisfactory performance. In addition, similar to Onion, PREFER is only suitable for static data. The performance of Onion can be improved with the use of indexing [23]; however, building robust indexes is quite expensive.

BRS [3] is a branch-and-bound approach for answering 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 [3], BRS is an I/O optimal algorithm, meaning 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 [24]. 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 [24] 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$10K) of users, while the group size in [24] is very small ($<$10). The proposed solution in [24] is obviously inapplicable to our problem.

8.2 Other related queries

There is plenty of work on skyline evaluation (e.g., [4], [5], [25]). 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 [4]. Papadias et al. [5] 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 [25].

Several new queries were proposed recently to assist the analysis tasks of product manufacturers. [26] uses the concept of dominance for business analysis from a microeconomic perspective, proposing a data cube model (DADA) to summarize the dominance relationships between objects in all combinations of dimensions. The space is modeled by the grid of dimensional value combinations (assuming that features have small integer domains) and each cell summarizes the dominance of products in it. In [6], the problem of creating competitive products have been studied. In [7], the authors aim at finding the best sub-space for a query object where it is highly ranked. Miah et al. [27] 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.

9 Conclusion

In this paper, we studied the problem of batch evaluation of numerous top-k queries. 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 *view-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. Moreover, we intend to study additional queries that can make use of $ATOP^k$ as a module.

ACKNOWLEDGMENT

This work was supported by grant HKU 715711E from Hong Kong RGC.

REFERENCES

[1] S. Chaudhuri and L. Gravano, "Evaluating Top-k Selection Queries," in *VLDB*, 1999, pp. 397–410.
[2] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," in *PODS*, 2001.
[3] Y. Tao, D. Papadias, V. Hristidis, and Y. Papakonstantinou, "Branch-and-Bound Processing of Ranked Queries," *Information Systems*, vol. 32, pp. 424–445, 2007.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *ICDE*, 2001, pp. 421–430.
[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[6] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating Competitive Products," *PVLDB*, vol. 2, no. 1, pp. 898–909, 2009.
[7] T. Wu, D. Xin, Q. Mei, and J. Han, "Promotion Analysis in Multi-Dimensional Space," *PVLDB*, vol. 2, no. 1, pp. 109–120, 2009.
[8] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Monochromatic and bichromatic reverse top-k queries," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 8, pp. 1215–1229, 2011.
[9] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis, "Identifying the Most Influential Data Objects with Reverse Top-k Queries," *PVLDB*, vol. 3, no. 1, pp. 364–372, 2010.
[10] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering Top-k Queries Using Views," in *VLDB*, 2006, pp. 451–462.
[11] Retrevo Survey http://www.retrevo.com/content/blog/2010/11/holiday-shopping-trends-and-black-friday-special-report.
[12] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-Nearest-Neighbors Queries in Spatial Databases," in *SSDBM*, 2004, pp. 297–306.
[13] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," in *SIGMOD Conference*. ACM Press, 1990, pp. 322–331.
[14] T. Bially, "Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction," *IEEE Transactions on Information Theory*, vol. 15, no. 6, pp. 658–664, 1969.
[15] A. Y. Halevy, "Answering Queries Using Views: A Survey," *VLDB J.*, vol. 10, no. 4, pp. 270–294, 2001.
[16] A. Packer, "NP-Hardness of Largest Contained and Smallest Containing Simplices for V- and H-Polytopes," *Discrete & Computational Geometry*, vol. 28, no. 3, pp. 349–377, 2002.
[17] J. Munkres, *Elements of Algebraic Topology*, 2nd ed. Prentice Hall, Jan. 1984, ch. 1.1.
[18] NBA Basketball Statistics http://www.databasebasketball.com/.
[19] Household dataset http://www.ipums.org/.
[20] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The Onion Technique: Indexing for Linear Optimization Queries," in *SIGMOD Conference*, 2000, pp. 391–402.
[21] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries," in *SIGMOD Conference*, 2001, pp. 259–270.
[22] V. Hristidis and Y. Papakonstantinou, "Algorithms and Applications for Answering Ranked Queries Using Ranked Views," *VLDB J.*, vol. 13, no. 1, pp. 49–70, 2004.
[23] D. Xin, C. Chen, and J. Han, "Towards Robust Indexing for Ranked Queries," in *VLDB*, 2006, pp. 235–246.
[24] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu, "Group Recommendation: Semantics and Efficiency," *PVLDB*, vol. 2, no. 1, pp. 754–765, 2009.
[25] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable Skyline Computation Using Object-Based Space Partitioning," in *SIGMOD Conference*, 2009, pp. 483–494.
[26] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang, "DADA: A Data Cube for Dominant Relationship Analysis," in *SIGMOD Conference*, 2006, pp. 659–670.
[27] M. Miah, G. Das, V. Hristidis, and H. Mannila, "Standing Out in a Crowd: Selecting Attributes for Maximum Visibility," in *ICDE*, 2008, pp. 356–365.

Shen Ge received his Bachelor's and Master's Degree in Computer Science from the Department of Computer Science and Technology in Nanjing University, China, in 2005 and 2008, respectively. He is currently a PhD candidate at the Department of Computer Science, University of Hong Kong, under the supervision of Prof. Nikos Mamoulis. His research focuses on query processing on multidimensional and spatial-textual data.

Leong Hou U received his Bachelor Degree in Computer Science and Information Engineering in 2003 from the National Chi Nan University, Taiwan, and received his Master Degree in E-Commerce in 2005 from the University of Macau, Macau. He received his PhD Degree in Computer Science from the University of Hong Kong in 2010. He is currently an Assistant Professor at the University of Macau. His research interest includes spatial and spatio-temporal databases, advanced query processing, web data management, information retrieval, data mining and optimization problems.

Nikos Mamoulis received a diploma in Computer Engineering and Informatics in 1995 from the University of Patras, Greece, and a PhD in Computer Science in 2000 from the Hong Kong University of Science and Technology. He is currently a professor at the Department of Computer Science, University of Hong Kong, which he joined in 2001. His research focuses on management and mining of complex data types, privacy and security in databases, and uncertain data management. He served as PC member in more than 80 international conferences on data management and mining. He is an associate editor for IEEE TKDE and the VLDB Journal.

David W. Cheung received the M.Sc. and Ph.D. degrees in computer science from Simon Fraser University, Canada, in 1985 and 1989, respectively. Since 1994, he has been a faculty member of the Department of Computer Science in The University of Hong Kong. His research interests include database, data mining, database security and privacy. Dr. Cheung was the Program Committee Chairman of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2001), Program Co-Chair of PAKDD 2005, Conference Chair of PAKDD 2007, and the Conference Co-Chair of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009).