Promotion Analysis in Multi-Dimensional Space

Tianyi Wu† Dong Xin‡ Qiaozhu Mei† Jiawei Han†
† University of Illinois at Urbana-Champaign
‡ Microsoft Research
{twu5,qmei2,hanj}@illinois.edu dongxin@microsoft.com

ABSTRACT

Promotion is one of the key ingredients in marketing. It is often desirable to find merit in an object (*e.g.*, product, person, organization, or service) and promote it in an appropriate community. In this paper, we propose a novel functionality, called *promotion analysis through ranking*, for promoting a given object by leveraging highly ranked results. Since the object may not be highly ranked in the global space, our goal is to discover *promotive* subspaces in which the object becomes prominent. To achieve this goal, the notion of *promotiveness* is formulated. We show that this functionality is practical and useful in a wide variety of applications such as business intelligence. However, computing promotive subspaces is challenging due to the explosion of search space and high aggregation cost. For efficient computation, we propose a PromoRank framework, and develop three efficient optimization techniques, namely subspace pruning, object pruning, and promotion cube, which are seamlessly integrated into the framework. Our empirical evaluation on two real data sets confirms the effectiveness of promotion analysis, and that our proposed algorithms significantly outperform baseline solutions.

1. INTRODUCTION

Promotion has been playing a key role in marketing. It is always desirable to identify competitive strengths of a product and promote it on the market. Can we systematically develop such a function that automatically finds interesting subspaces, if any, for a given object so that it can be promoted confidently?

In this paper, we present a novel functionality, called *promotion analysis through ranking*, that is important to many applications like business intelligence. In a nutshell, we exploit a common fact that ranking information, in particular top ranks, of a target object (*e.g.*, product, person, business, organization, or service) can serve as effective means for promotional purposes. For example, "Fortune 500" could deliver a positive image of an enterprise to customers and be used for promotion, and a basketball player described by "scoring champion" can easily impress others as well. Therefore, *our goal in this paper is to leverage such interesting ranking information for object promotion*. We observe that in many cases such information would be simply impossible to obtain, as most objects may not be prominent globally; that is, when comparing to *all* competitors in *all* aspects. Thus, toward our goal, subspace ranking analysis is carried out to discover interesting patterns of object ranking in subspaces.

Example 1. (PRODUCT PROMOTION) *A book retailer manager intends to promote their brand. Unfortunately, in terms of book sales, they rank lower than 30-th among all book retailers. However, when breaking down the market into segments, such as* Year*,* Category*, and* Readership*, she finds out that they are in fact* the top-1 bookseller in the {Readership = College Students, Category = Science and Technology} segment*. This information can be effectively used for making advertisement and allocating marketing resources to seek profits.*

Example 2. (PERSON PROMOTION) *An NBA manager would like to promote Michael Jordan as a superstar. Having checked the statistics, he realizes that Jordan is only ranked as the 3rd all-time leading scorer. However, further analysis suggests more exciting results:* Jordan is the top scorer in the guard position, the top scorer on the Chicago Bulls team, as well as 11 individual years' scoring champion.

These examples show typical applications of promotion. The strategy adopted here is to break down the data space into subspaces so that a globally low-ranked object becomes prominent in some subspaces, which can be then used for promotion. In fact, this strategy is not new as it has been extensively studied and practiced in marketing [13]. While existing commercial database and data warehouse systems can well support the functionality of "retrieving top objects in some subspace", there exists no subspace ranking analysis method for promotional purposes. It would be prohibitively difficult for users to navigate large data sets, because of the potentially exponential number of subspaces as well as the high cost of aggregation and ranking. In this paper, we systematically study this problem, which, to the best of our knowledge, has not been studied before.

(PROMOTION QUERY) *Given a target object, our goal is to discover its top-R promotive subspaces.*

To address the problem, a new notion called *promotiveness* need to be formulated. Intuitively, that a target object ranks high in a subspace suggests that the subspace is *promotive*. The promotion query returns the target object's R most promotive subspaces, which may have different uses depending on the task at hand. First, the results can be directly used to find merit in the target object, raising its image and profile. Second, it is able to help analysts discover the right market segment for promotion. For example, if a book is highly ranked among college students, one would allocate

VLDB '09, August 24-28, 2009, Lyon, France

Location	Year	Object	Score
NY	2008	T_1	0.5
WA	2008	T_1	0.8
WA	2007	T_2	1.0
WA	2008	T_2	1.0
NY	2007	T_3	0.3
WA	2007	T_3	0.6
WA	2008	T_3	0.7

Figure 1: Sample multidimensional data.

Subspace	Rank	ObjCount
{*}	3rd	3
{NY}	1st	2
{WA}	3rd	3
{2008}	1st	3
{NY, 2008}	1st	1
{WA, 2008}	2nd	3

Figure 2: Target object T_1's subspaces and its ranks.

the marketing resources on colleges. Third, high rankings in promotive subspaces can be more meaningful and informative to an information seeker. For example, Forbes and U.S. News and World Report regularly publish ranked results of businesses, universities, not only in general, but also in various subfields (*e.g.*, undergraduate vs. graduate education). A university may discover "ranked top-3 in biomedical research" to be more useful than "ranked top-15 in medical sciences as a whole".

Example 3. (CONTEXT) *We use the multidimensional model to accommodate typical decision support data. Figure 1 illustrates a sample fact table, where dimensions are categorized by subspace dimensions (*Location, Year*), object dimension (*Object*), and score dimension (*Score*). Assume that our target object for promotion is T_1. Figure 2 lists T_1's 6 subspaces and the corresponding rank of T_1 in each subspace. Here assume that the rank is derived by ordering objects in a subspace in descending order according to the* SUM *aggregate score. For instance, in $\{WA\}$ T_1 ranks 3rd because $T_2 : 2.0 > T_3 : 1.3 > T_1 : 0.8$.*

In different contexts, two types of object rankings can be distinguished. (i) *Aggregate score-based*: the computation of an object's rank in any subspace involves dynamic score aggregation (such as in the above example); (ii) *Constant score-based*: each object's score is constant across all subspaces, so objects' relative ranking will be fixed in any subspace. We focus on the former case, which is more general.

In order to quantify how well a subspace can serve a target object for promotion, we need the key notion of promotiveness. Intuitively, a subspace in which the target object is highly ranked would have high promotional value. However, this may not be the whole story.

Example 4. (PROMOTIVENESS) *Continue with our running example in Figure 2. Observe that, $\{2008\}$ is a promotive subspace because T_1 ranks 1st in it, much better for promotion than the full space $\{*\}$, where T_1 ranks last. However, also observe that, although T_1 has equal ranks in $\{2008\}$ and $\{NY\}$, these two subspaces may not be considered equally promotive, because in $\{2008\}$ T_1 has two competitors but in $\{NY\}$ there is only one. Clearly, this indicates "more competition" in $\{2008\}$ and it is desirable that $\{2008\}$ be considered more promotive. For the same reason, $\{NY, 2008\}$ may not be considered promotive even though T_1 ranks 1st.*

That the target object is highly ranked in a subspace does not necessarily suggest that the subspace is promotive, because other factors such as the number of competitors would affect its promotional value as well. In this paper, we support a class of measures to gauge the promotiveness of subspace. Instead of solely relying on rank, another element coined *subspace significance* is considered. This class of measures enables users to model context-specific semantics. However, a potential *spurious promotion* problem may arise when seemingly promotive subspaces are actually caused by random noises. To tackle the problem, statistical methods based on analysis of variance are proposed as an integral part of our solution.

Technical challenges: The promotion query problem presents significant challenges because of the following reasons. First, in a d-dimensional data set, there is potentially an exponential number of subspaces. A naive approach that enumerates all subspaces and computes the promotiveness value for each subspace would be prohibitively expensive. Second, the promotiveness measure is neither monotonic nor anti-monotonic. In other words, the promotiveness of a child subspace can be either higher or lower than that of its parent subspace. For example, in Figure 2, subspace {2008} may have a higher promotiveness than both its child subspace {NY, 2008} and its parent subspace {*}. Such non-monotonicity prevents us from utilizing existing aggregate computation methods, which require monotonicity of the measure. Third, the promotiveness measure is holistic [10], which means that computing the promotiveness measure incurs high aggregation cost, as scores must be aggregated for all objects in order to derive the target object's rank. This makes shared computation of the promotiveness measure across subspace lattice difficult.

While answering promotion query is challenging, we develop algorithms that significantly outperform baseline solutions, making promotion analysis feasible for large-scale applications. We first propose a generic PromoRank framework, and then develop the following optimization techniques by exploiting the fact that users are only interested in the *top* subspaces where the target object has *top* ranks. (i) *Subspace pruning*: Users only desire the most promotive subspaces, so aggregations in many "unpromising" subspaces could be wasted. To avoid unnecessary aggregations, we establish upper and lower bounds for the promotiveness measure by utilizing the parent-child relationships among subspaces. The results of subspaces which have already been aggregated may be reused to prune unseen ones with little overhead. (ii) *Object pruning*: In many real data sets, object scores follow power-law distributions. When computing a target object's rank in a subspace, only objects with aggregate score larger than that of the target object would affect the target object's rank, whereas objects in the long tail would not. Therefore, they can be chopped off at an early stage, thereby reducing subsequent aggregation and ranking cost. (iii) *Promotion cube*: We develop a compact materialization strategy to further optimize the efficiency of query processing, which complements the online algorithms and seeks the middle ground between space overhead and query execution time. All these techniques are seamlessly integrated in our framework. The contributions of this paper are summarized as follows.

- Present the promotion analysis problem and its uses. To the best of our knowledge, this is the first paper to systematically study the problem;
- The notion of subspace promotiveness is formalized (Section 2);

- Efficient query execution algorithms are proposed in the PromoRank framework (Section 3); a compact cube structure, called *promotion cube*, is proposed to further speedup query processing (Section 4);
- Discuss statistical methods for preventing spuriously promotive results (Section 5);
- Verify the quality of promotion analysis using two real-world data sets. An extensive performance study shows that our proposed techniques are much more efficient than baseline algorithms (Section 6).

The remainder of this paper is organized as follows. Section 2 formally introduces the promotiveness measure and the problem definition. Section 3 discusses the PromoRank framework and the pruning techniques. Section 4 proposes the promotion cube. Section 5 discusses the method for removing spuriously promotive subspaces. Section 6 reports our experimental results. Section 7 reviews related work. Finally, Section 8 concludes this study.

2. PROBLEM DEFINITION

2.1 Data Model

Consider a d-dimensional data set $\mathcal{D}$ consisting of a set of n base tuples, each having d (categorical) **subspace dimensions** $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$, an **object dimension** I_{obj}, and a **score dimension** I_{score}. Denote $dom(I_{obj})$ by $\mathcal{O}$, namely the complete set of object IDs (*e.g.*, in Example 3, $\mathcal{O} = \{T_1, T_2, T_3\}$). Let $dom(I_{score})$ be $\mathbb{R}^+$, the set of non-negative real numbers.

A **subspace** is defined as $S = \{a_1, a_2, \ldots, a_d\}$, where $a_i \in A_i$ or $a_i = *$ (star refers to the "any" value). S induces a projection of the data set $\mathcal{D}_S (\subseteq \mathcal{D})$ and a subspace of objects $\mathcal{O}_S (\subseteq \mathcal{O})$ (*e.g.*, in Example 3, $\mathcal{O}_{\{NY\}} = \{T_1, T_3\}$). A subspace $S_1 = \{a_1, a_2, \ldots, a_d\}$ is called a **child** subspace of $S_2 = \{b_1, b_2, \ldots, b_d\}$ iff there exists a j s.t. $a_j \neq * \wedge b_j = *$ and $a_i = b_i$ for any $i \neq j$. Conversely, S_2 is a **parent** subspace of S_1.

For this d-dimensional data, all subspaces can be partitioned into 2^d **cuboids** (or group-by's). We say that S belongs to a d'-dimensional cuboid $\mathcal{A}'$ denoted by $A'_1 A'_2 \cdots A'_{d'}$ iff S has non-star values in these d' dimensions and star values in the other $d - d'$ dimensions. These 2^d cuboids form a cuboid lattice, where in particular the apex cuboid denoted by "$*$" contains only the full space $\{*, *, \ldots, *\}$.

In subsequent discussions, assume a query **target object** $t_q \in \mathcal{O}$ is given. Let $\mathbf{S}_q = \{S_q | t_q \in \mathcal{O}_{S_q}\}$ be the set of **target subspaces** where t_q occurs. These subspaces form a **target lattice**. In Example 3, the target object T_1 has 6 target subspaces as shown in Figure 2; subspace $\{2007\}$ is not a target subspace because T_1 does not occur in it.

2.2 A Unified Promotiveness Measure

For any target subspace S_q, we need to measure its promotional value for t_q. We assume that a higher rank of t_q in S_q should make S_q promotive. On the other hand, fixing t_q's rank, more competitors in S_q indicates that it has larger promotional value. To formalize the intuition, we define *promotiveness* as a class of composite measures.

Definition 1. (PROMOTIVENESS) *Given t_q and S_q, the promotiveness,* P*, can be defined as*

$$\mathsf{P}(t_q, S_q) = \mathsf{f}\big(\mathsf{Rank}(t_q, S_q)\big) \cdot \mathsf{g}\big(\mathsf{Sig}(S_q)\big), \quad (1)$$

where Rank *measures the rank of t_q in S_q based on a given monotone aggregate measure $\mathbb{M}$,* Sig *measures S_q's own* significance *(i.e., promotional value), and* f *and* g *are monotone normalization functions.*

P consists of three components: Rank, Sig, and f and g. We elaborate on each component and its assumptions in the following.

2.2.1 The Rank measure

Definition 2. (RANK) *Given an aggregate measure $\mathbb{M}$, t_q's rank in S_q is defined as*

$$\mathsf{Rank} = \big|\{t | t \in \mathcal{O}_S \wedge t \neq t_q \wedge M^S(t) \rhd M^S(t_q)\}\big| + 1, \quad (2)$$

where $\rhd \in \{>, \geq, <, \leq\}$ and $M^S(t)$ denotes the aggregate score of object t in subspace S.

Rank is a deciding factor in promotiveness. Assume that $\mathbb{M}$ is a monotone measure such as *SUM*, *COUNT*, *MAX*, *etc.*. For simplicity, also assume $\rhd =$ " $>$ ", and our techniques can be extended to other relation operators.

2.2.2 The Sig measure

Sig measures the promotional value of the subspace itself, which is independent of t_q. Some instances of this measure are TupleCount (the number of base tuples), ObjCount (the number of distinct objects), or Level (the number of star-values). For example, a larger ObjCount may suggest that the subspace is "more competitive". In general, assume Sig is monotone such that the Sig of a child subspace should be no greater than that of its parent subspace.

2.2.3 The normalization functions f and g

$\mathsf{f}(\cdot)$ and $\mathsf{g}(\cdot)$ are monotone normalization functions that combine Rank and Sig in order to derive a meaningful promotiveness value. We require $\mathsf{r}_1 \leq \mathsf{r}_2 \Rightarrow \mathsf{f}(\mathsf{r}_1) \geq \mathsf{f}(\mathsf{r}_2)$ and $\mathsf{s}_1 \leq \mathsf{s}_2 \Rightarrow \mathsf{g}(\mathsf{s}_1) \leq \mathsf{g}(\mathsf{s}_2)$.

2.2.4 Example instantiations

We illustrate several example instantiations of P to model certain semantics of promotion query.

Enforcing iceberg constraint: One might use Rank to gauge the promotiveness of subspaces while enforcing a minimum support threshold *minsup* to filter out "small" subspaces. In this case, P could be like

$$\mathsf{P} = \mathsf{f}(\mathsf{Rank}) \cdot \mathcal{I}(\mathsf{TupleCount} \geq \mathit{minsup}),$$

where $\mathcal{I}$ is an indicator function (which returns 1 when the condition is true and 0 otherwise) and f is any monotone function. Thus any subspace not passing *minsup* will have 0 promotiveness.

Percentile rank: Another useful instantiation of P is the percentile rank. For example, a subspace where t_q ranks to top-1% might be much more promotive than a subspace with percentile rank top-30%. The percentile rank is a common measurement for studying student test scores. Formally, we can instantiate P as

$$\mathsf{P} = \mathsf{Rank}^{-1} \cdot \mathsf{ObjCount},$$

such that a higher percentile rank of t_q would result in a larger promotiveness value.

Other functions: Users may propose other functions to customize P. For example, one may use *continuous* g function to penalize "small" subspaces, or assign static *weights* to subspaces (*e.g.*, larger weights are associated with the recent years than with the past).

2.3 The Promotion Query Problem

Definition 3. (PROMOTION QUERY) *Given data set $\mathcal{D}$, target object t_q, and promotiveness measure* P*, return the top-R subspaces with the largest promotiveness values.*

Ties are broken arbitrarily. For clarity of presentation, in the subsequent discussions, assume $\mathbb{M}$ is *SUM* and let P be $\mathsf{Rank}^{-1} \cdot \mathcal{I}(\mathsf{TupleCount} \geq \textit{minsup})$. These assumptions do not make our solutions less general to the query model. We discuss how to avoid spuriously promotive subspaces generated by random noise in Section 5.

3. THE PROMORANK FRAMEWORK

In this section we discuss the PromoRank framework with two pruning methods: *subspace pruning* and *object pruning*. We start with a general framework that lays the foundation. This framework is based on the bottom-up computation method discussed in [5]. The general idea is as follows. It runs in memory and recursively partitions the data set according to some dimension, and objects are aggregated in the subspace corresponding to each partition. Then the promotiveness measure value for that subspace is computed. Table 1 displays the outline of the framework, PromoRank, based on the example instantiation of promotiveness measure assumed in Section 2.3, *i.e.*, $\mathsf{P} = \mathsf{Rank}^{-1} \cdot \mathcal{I}(\mathsf{TupleCount} \geq \textit{minsup})$. The framework is divided into an aggregation phase and a partition phase.

Aggregation phase: In this phase, the Rank and P measures are computed for the input subspace S (Line 2). Then the target subspace S and its P value will be inserted into a priority queue which maintains the top-R results (Line 3). We now elaborate on the computation of P for S: Scan through the I_{obj} and I_{score} dimensions of the input base tuples in $\mathcal{D}$ and compute the aggregate score (*i.e.*, *SUM*) for each object in $\mathcal{O}$. This is implemented using a hash table keyed on object ID. The resulting hash table would have $|\mathcal{O}|$ entries which map each object to its aggregate score. Rank is computed directly by counting the number of aggregate scores strictly larger than t_q's aggregate score. Finally, P is set to Rank^{-1}.

Partition phase: The input data $\mathcal{D}$ is iteratively sorted according to the d'-th dimension in a depth-first search manner (Line 5). As a result, $\mathcal{D}$ can be projected into multiple partitions such that each partition corresponds to a distinct value on the d'-th dimension. A child subspace S' of S is defined on each partition (Line 7), and PromoRank recursively progresses over subspace S' and the corresponding data partition of S' (Line 8). Any subspace where the target object does not occur will be pruned (Line 1). Also, the iceberg constraint "$\mathsf{TupleCount} \geq \textit{minsup}$" can be enforced as well (Line 1). Figure 3 displays an example recursive process for 4 dimensions at cuboid-level. For each cuboid, we label the order it is visited by PromoRank.

Algorithm 1: PromoRank(S, $\mathcal{D}$, $\mathcal{O}$, d_0)	
Input	
Target object t_q, subspace S, and data set $\mathcal{D}$;	
Object set in current subspace $\mathcal{O}$;	
Previous partition dimension d_0.	
Output:	
Top-R promotive subspaces *Result*.	
1	**if** $\lvert\mathcal{D}\rvert < \textit{minsup} \vee t_q \notin \mathcal{O}$ **then return;**
2	Compute Rank and P;
3	Update *Results* using (S, P);
4	**for** $d' \leftarrow d_0 + 1$ **to** d **do**
5	Sort $\mathcal{D}$ based on d'-th dimension;
6	**foreach** value v in d'-th dimension **do**
7	$S' \leftarrow S \cup \{d' : v\}$;
8	PromoRank(S', $\mathcal{D}_{S'}$, $\mathcal{O}_{S'}$, d');
9	**end**
10	**end**

Table 1: The PromoRank algorithm.

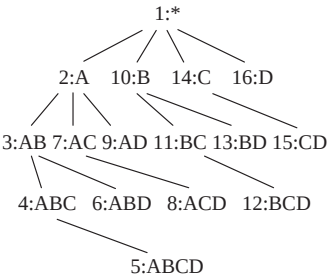


Figure 3: Cuboid tree.

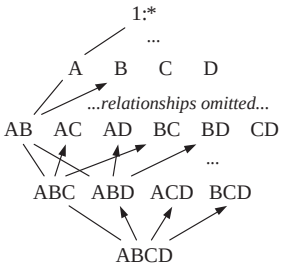


Figure 4: Using subspace relationships to lower bound Rank.

Although the PromoRank algorithm presented here supports an instance of the promotiveness measure with iceberg constraint, it is nevertheless generic that it is able to compute *any* promotiveness measure. Specifically, given any aggregate measure $\mathbb{M}$, Rank can be computed using the hash table in the same way as described above, and Sig can be directly derived from the input subspace S and/or the input data $\mathcal{D}$.

Analysis: At each recursion the aggregation runs in $O(|\mathcal{D}| + |\mathcal{O}|)$ time, due to the scan of the input data and hashing on objects. The partition phase runs in $O(|\mathcal{D}|)$ time because the input data was sorted during the last recursion. Overall there will be $|\mathbf{S}_q|$ recursions, so the total cost of the algorithm can be written as $C^{all} = \sum_{i=1}^{|\mathbf{S}_q|}(C_i^{par} + C_i^{agg})$, where C_i^{par} and C_i^{agg} are the partition and aggregation costs at the i-th recursion, respectively.

Notice that this algorithm computes *all* subspaces, and thus the overall cost C^{all} could be quite prohibitive for large data sets. Because users often ask for only the *top* subspaces where the target object has *top* ranks, we further develop optimization techniques in the subsequent sections.

3.1 Subspace Pruning

Now we discuss the subspace pruning technique. The key motivations for this technique are that (i) users are often only interested in top-R promotive subspaces, and (ii) the aggregate measure $\mathbb{M}$ is monotone (*e.g.*, *SUM*). Intuitively, it could be wasteful to perform aggregation for all subspaces. To prune out "unpromising" subspaces, the problem becomes to establish an upper bound for the promotiveness measure.

Given the set of target subspaces $\mathbf{S}_q$, the PromoRank algorithm iterates through each subspace in it in a sequential order, $S_1, S_2, \ldots, S_{|\mathbf{S}_q|}$ (illustrated in Figure 3), if both the dimension ordering and the value ordering on each dimension are fixed. At any time of the algorithm, the sequence of subspaces can be conceptually split into a list of seen subspaces which have already been aggregated, and a list of unseen subspaces which have not yet been aggregated. From the list of seen subspaces, we compute the current R-th largest promotiveness value as a threshold. For the list of unseen subspaces, we derive an upper bound promotiveness value for each of them using already aggregated results. Thus, any un-

seen subspace whose upper bound is less than the threshold can be pruned. To derive the upper bound, we utilize parent-child relationships between seen and unseen subspaces in the target lattice.

We first introduce some notation. At any time k of the algorithm, let

- $\mathbf{S}_{seen}$ denote the *list of seen subspaces* $\{S_1, \ldots, S_k\}$ $(0 \leq k \leq |\mathbf{S}_q|)$ which have been aggregated or pruned already;
- $\mathbf{S}_{unseen}$ denote the *unseen subspace list* $\{S_{k+1}, \ldots, S_{|\mathbf{S}_q|}\}$;
- $M^i(t)$ denote the *aggregate score* of object t $(\in \mathcal{O}_{S_i})$ in any subspace S_i $(1 \leq i \leq |\mathbf{S}_q|)$; in particular, $M^i(t_q)$ denote the aggregate score of the target object in S_i;
- $S_i.\mathsf{P}$, $S_i.\mathsf{Sig}$, and $S_i.\mathsf{Rank}$ denote the *exact measure values* of S_i $(1 \leq i \leq |\mathbf{S}_q|)$;
- $S_i.\overline{\mathsf{P}}$, $S_i.\overline{\mathsf{Sig}}$, and $S_i.\underline{\mathsf{Rank}}$ denote the *upper bound* P, *upper bound* Sig, and *lower bound* Rank of an unseen subspace S_i $(k+1 \leq i \leq |\mathbf{S}_q|)$;
- $\underline{\mathsf{P}}$ denote the *R-th largest promotiveness measure value of all seen subspaces*.

Now we consider the problem of computing a general upper bound for promotiveness, *i.e.*, computing $S_i.\overline{\mathsf{P}}$ given an unseen subspace S_i. By definition, $S_i.\overline{\mathsf{P}} = \mathsf{f}(S_i.\underline{\mathsf{Rank}}) \cdot \mathsf{g}(S_i.\overline{\mathsf{Sig}})$ because of the monotonicity of f and g. In general, $S_i.\overline{\mathsf{Sig}}$ can be computed based on S_i's seen parents since Sig is monotone (note that the iceberg constraint assumed is a specific instance of $\mathsf{g}(\mathsf{Sig})$). So now the problem is reduced to computing $S_i.\underline{\mathsf{Rank}}$. Because Rank is neither monotone nor convex, we cannot compute $S_i.\underline{\mathsf{Rank}}$ directly from S_i's parent subspaces. A trivial lower bound is 1, which assumes the best possible rank; however, this would not be able to provide any pruning power. Our idea here is to exploit the monotonicity of the aggregate measure in the subspace lattice.

Claim 1. (RANK MEASURE LOWER BOUND) *Let S_c be any child subspace of S, we can obtain a lower bound* Rank *measure as* $S.\mathsf{Rank} \geq |\{t | t \in \mathcal{O}_{S_c} \wedge M^{S_c}(t) > M^S(t_q)\}| + 1$.

The claim is clear as aggregate scores must be monotone across parent-child subspaces. Therefore, given an unseen subspace S_i and its seen child subspace S_j that has already been aggregated, we can compute the lower bound Rank for S_i as $S_i.\underline{\mathsf{Rank}} = \big|\{t | t \in \mathcal{O}_{S_j} \wedge M^j(t) > M^i(t_q)\}\big| + 1$, by using $M^j(t)$, namely the already aggregated object scores in the child subspace S_j, and $M^i(t_q)$, namely the target object's aggregate score in S_i. When there are multiple unseen child subspaces of S_i, there could be multiple lower bounds. In such cases, the maximum value is taken to provide the tightest bound. This way, the upper bound for promotiveness can be established, enabling our subspace pruning strategy.

Algorithm 2: PromoRank+

```
1  for i ← 1 to |S_q| do /* initialization */
2      Compute M^i(t_q);
3      S_i.P̄ = S_i.Siḡ ← ∞, S_i.Rank̲ ← 1;
4  end
5  P̲ ← 0;
6  D̄ = D\{t_q};
7  for i ← 1 to |S_q| do
8      if S_i.P̄ > P̲ then /* if not pruned */
9          Compute S_i.P, update P̲ and top-R results;
10         foreach S_i's descendent S_j (j > i) do
11             S_j.Siḡ ← S_i.Sig;
12             Update S_j.P̄;
13         foreach S_i's parent S_j (j > i) do
14             Update S_j.Rank̲ and S_j.P̄;
15     end
16     Sort and partition; /* recursive */
17 end
```

Table 2: The subspace pruning algorithm.

PromoRank+, the subspace pruning algorithm based on the basic framework, is displayed in Table 2. However, as shown in Table 2, it is "flattened" for simplicity (*i.e.*, instead of showing a recursive algorithm, we use a for-loop to represent it). Note that in order to compute $S_i.\underline{\mathsf{Rank}}$ for any S_i, we need to first compute the target object's aggregate score in each target subspace (Line 2). Also, all bounds are initialized (Lines 3 and 5), and the base tuples pertaining to the target object are removed from the data (Line 6). Here computing $M^i(t_q)$ for all target subspaces is very efficient because the number of base tuples related to t_q often makes only a small portion of the data. This could be even more efficient if a clustered index has been built on the object dimension.

During partitioning, PromoRank+ iteratively aggregates each subspace (Lines 7–17). At each iteration, one of the following two cases happens. If the upper bound promotiveness of the current subspace is less than the R-th largest promotiveness value seen so far, the subspace can be pruned. In this case, its aggregation step can be avoided (*i.e.*, skips Line 9–14). Otherwise, object score aggregation must be performed to obtain the exact P measure value (Line 9). After that, $S_i.\mathsf{Sig}$ can be used to upper bound the Sig measure of all S_i's descendent subspaces (Lines 10–11). Meanwhile, the resulting aggregate scores can be reused to derive $\underline{\mathsf{Rank}}$ for its unseen parent subspaces (Lines 13–14). The updated bounds of Sig and Rank of unseen subspaces are then propagated to the upper bound of P (Lines 12 and 14).

Example 5. Figure 4 displays the computation of lower bound Rank *in a lattice view. Subspaces are recursively aggregated in the order $A \rightarrow \cdots \rightarrow ABCD \rightarrow ABD \rightarrow \cdots$. When, for example, a subspace in ABC has just been aggregated, the* $\underline{\mathsf{Rank}}$ *of its corresponding unseen parents in AC and BC can be updated; we do not update AB because it is seen. Similarly, aggregated results in* ABD *can be reused for AD and BD.*

Figures 5 and 6 further show a concrete example. Assume the target object is t_7. Also assume $\mathsf{P} = \mathsf{Rank}^{-1}$. When PromoRank *is executed, 4 target subspaces will be aggregated in the order $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$. The aggregate scores in each subspace are listed in Figure 6, from which* Rank *and* P *can be derived.*

Now suppose PromoRank+ *is executed on the same data and $S_1 \rightarrow S_2 \rightarrow S_3$ have just been aggregated. Because S_3 has an unseen parent subspace S_4, we can reuse S_3's result to compute $S_4.\underline{\mathsf{Rank}} = \big|\{t | t \in \mathcal{O}_{S_3} \wedge M^3(t) > M^4(t_q)\}\big| + 1$. Since we have $M^4(t_q) = 0.4$ during initialization, $S_4.\underline{\mathsf{Rank}} = |\{t_3, t_6\}| + 1 = 3$. In this case, if we want to find the top-1 promotive subspace, we would have $\underline{\mathsf{P}} = S_2.\mathsf{P} = 1/2$ (i.e., 1/2 being the largest* P *seen so far), and we can safely prune out S_4 because $S_4.\overline{\mathsf{P}} = 1/4 < \underline{\mathsf{P}}$.*

Analysis: $S_i.\underline{\mathsf{Rank}}$ can provide effective pruning power when $M^i(t_q)$ is relatively small, since in such cases $S_i.\underline{\mathsf{Rank}}$ would be bounded away from top ranks. Because our goal is to find the top-R most promotive subspaces, many target subspaces with small $M^i(t_q)$ can be pruned, leading to lower aggregation cost. Another advantage of subspace pruning is that subspaces are pruned with little overhead: First, computing t_q's aggregate scores beforehand incurs no redundant cost because all base tuples related to t_q are removed subsequently. Second, no intermediate aggregate results

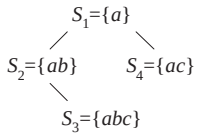


Figure 5: A subtree of subspaces rooted at S_1.

Subspace	Objects ($\mathcal{O}_S$) and their aggregate scores ($M^S(t)$)	$M^S(t_q)$	Rank	P
$S_1 = \{a\}$	$t_6(1.2)\ t_3(1.0)\ t_1(0.7)\ t_7(0.7)\ t_4(0.3)\ t_5(0.3)\ t_2(0.2)$	0.7	3	1/3
$S_2 = \{ab\}$	$t_6(0.7)\ t_3(0.6)\ t_7(0.6)\ t_1(0.4)\ t_4(0.3)\ t_2(0.2)\ t_5(0.2)$	0.6	2	1/2
$S_3 = \{abc\}$	$t_3(0.6)\ t_6(0.5)\ t_7(0.3)\ t_1(0.1)\ t_5(0.1)$	0.3	3	1/3
$S_4 = \{ac\}$	$t_3(0.8)\ t_6(0.7)\ t_1(0.6)\ t_7(0.4)\ t_5(0.2)\ t_2(0.1)\ t_4(0.1)$	0.4	4	1/4

Figure 6: Example subspaces and their aggregated results.

need to be stored because the lower bound Rank is computed during aggregation.

3.2 Object Pruning

In the basic PromoRank framework, the Rank measure is computed in a holistic manner. That is, for each subspace, the complete set of objects in that subspace are aggregated and compared in order to derive Rank for the target object. However, this often incurs huge waste as only the objects which have larger aggregate scores than that of the target object would determine Rank, regardless of how many objects there are. We motivate this pruning technique using a typical real-world example: Figure 7 displays a power-law distribution in the DBLP data set [1], where the X-axis (object) corresponds to more than $450K$ authors and the Y-axis (aggregate score) represents the number of publications each author has. We observe that most authors in the long tail would not affect the computation of Rank, and thus performing aggregation at each recursion for *all* objects and recursively passing these objects to the next child subspace would be unnecessary. This motivates us to develop the object pruning technique, which aims to determine the *minimal set* of objects that affect Rank, so that all remaining objects that do not fall into this set are pruned early.

For a target subspace S_i, let S_i.*MinScore* denote the minimum aggregate score of the target object in the S_i's subtree. Here the *subtree* is defined as the depth-first search tree induced by the recursive process, as illustrated in Figure 3. For example, in Figures 5 and 6, S_1's subtree consists of S_1 through S_4, and S_1.*MinScore* $= \min\{0.7, 0.6, 0.3, 0.4\} = 0.3$.

Therefore, given a subspace S_i and any object $t \in \mathcal{O}_{S_i}$, t can be pruned if $M^i(t) \leq S_i$.*MinScore*. This is because, for any subspace S_j in S_i's subtree, $M^j(t) \leq M^i(t) \leq S_i$.*MinScore* $\leq M^j(t_q)$, which means that t's aggregate score is no greater than the target object's aggregate score in S_j. In other words, t would not affect S_j.Rank, and this holds true for any S_j in S_i's subtree during the depth-first search process. Therefore, t can be pruned from $\mathcal{O}_{S_i}$. Once t is pruned, its corresponding base tuples in the current data partition $\mathcal{D}_{S_i}$ can be pruned as well.

Example 6. Continue with the example in Figures 5 and 6, where S_1.MinScore=0.3. Suppose the aggregate scores of objects in $\mathcal{O}_{S_1}$ have just been computed for S_1. By checking these scores, one can see that t_4 (0.3), t_5 (0.3), and t_2 (0.2) have scores no greater than S_1.MinScore. This means that t_4, t_5, and t_2 would not affect Rank *of any subspace in S_1's subtree (i.e., S_2, S_3, and S_4). Therefore, they can be safely pruned so that they do not need to be aggregated in S_2, S_3, and S_4. In Figure 6, we can see that the* Rank *of t_7 in all 4 subspaces are not decided by t_4, t_5, or t_2.*

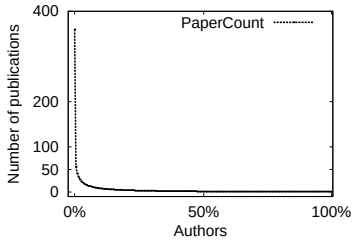


Figure 7: Aggregate score distribution in DBLP.

This object pruning strategy can be pushed down into the PromoRank algorithm recursively and be integrated with subspace pruning. The complete query execution algorithm, PromoRank++, is shown in Table 3. We highlight its differences from PromoRank+ as follows. (i) Initializing *MinScore*: This happens at the initialization stage (Line 5), which can be efficiently computed bottom-up in the cuboid tree. (ii) Recursive object pruning: At the end of each aggregation phase (except for those subspaces without any child subspace), identify the set of objects to be pruned $\mathcal{O}_{min} = \{t | t \in \mathcal{O}_S \wedge M^S(t) \leq S.MinScore\}$ (Line 16). Then, prune all base tuples related to $\mathcal{O}_{min}$ from the current data partition (Line 17).

Analysis: By pruning objects and data, the cost of both aggregation and partitioning would be greatly reduced. This pruning strategy introduces little overhead as $\mathcal{O}_{min}$ can be directly computed from aggregated results, and the additional space overhead is $O(1)$ per subspace. However, using this object pruning strategy, objects and data are pruned at early stage and we may not be able to compute some Sig measures like ObjCount exactly (other measures like Level remain unaffected). Two techniques may be used to address this problem. First, approximate Sig using selectivity estimation techniques like [18]. Because promotiveness is not sensitive to Sig, such approximation often would not harm result quality. In

Algorithm 3: PromoRank++

```
1  for i ← 1 to |S_q| do /* initialization */
2      S_i.P̄ = S_i.Siḡ ← ∞, S_i.Rank ← 1;
3      Compute M^i(t_q);
4  end
5  Compute S_i.MinScore for each S_i;
6  P ← 0;
7  D̄ ← D\{t_q};
Procedure: PromoRank++(S, D, O, d_0)
8  if S.P̄ > P then /* subspace pruning */
9      Compute S.Rank, S.Sig, and S.P;
10     Update P and top-R results;
11     foreach S' descendent S_j do
12         S_j.Siḡ ← S_i.Sig;
13         Update S_j.P̄;
14     foreach S' parent S_k (k > i) do
15         Update S_k.Rank and S_k.P̄;
16     Compute O_min using S.MinScore;
17     D ← D\O_min; /* object pruning */
18 end
19 for d' ← d_0 + 1 to d do /* partition */
20     Sort D based on d'-th dimension;
21     foreach child target subspace S' along d'-th dimension do
22         PromoRank++(S', D_S', O_S'\O_min, d');
23 end
```

Table 3: The complete query execution algorithm.

our experiments we verified that a simple approximation method could work very well. Second, one may materialize Sig of subspaces which pass a *minsig* threshold so that Sig can be accurately obtained.

4. PROMOTION CUBE

In practice, interactive and explorative analysis requires very short response time of queries. To further speedup promotion query processing for real-world applications, we propose a *Promotion Cube* technique to complement the online algorithms. The goal of the promotion cube is to (i) quickly locate promotive subspaces, and (ii) effectively prune out less promotive subspaces for an arbitrary target object. Toward this end, we exploit two types of correlations. First, the target object's rank is strongly correlated with its promotiveness value. In other words, high rank likely leads to large promotiveness. Second, a promotive subspace is unlikely to be insignificant. Therefore, the promotion cube precomputes only subspaces with Sig above a certain threshold *minsig*, which is similar to an iceberg cube. In each precomputed subspace, instead of directly materializing objects and their promotiveness values, we materialize a set of order statistics. Specifically, only a very small set of the largest aggregate scores is precomputed without considering actual object IDs. A key advantage of this structure is that the threshold *minsig* and the size of the order statistics do not limit the capability of query processing. Any top-R promotive subspace can be discovered for any target object even if the subspace does not satisfy the threshold or if the target object's aggregate score in that subspace is not precomputed.

The definition of the promotion cube structure is as follows. Consider data set $\mathcal{D}$ and aggregate measure $\mathbb{M}$. Assume two cube parameters, *maximum rank* k and *minimum significance threshold minsig*, are given. Another optional parameter, *cell size* k', will be discussed shortly.

Definition 4. (PROMOTION CELL) *Given a subspace S, a promotion cell $S.PCell = (M_i)_{i=1}^{k}$ is defined as the sequence of the top-k largest object aggregate scores in S.*

Definition 5. (PROMOTION CUBE) *The promotion cube $\mathbb{D}$ consists of a set of triples in the format $(S, PCell, \mathsf{Sig})$, where any subspace S must pass the* minsig *threshold. Formally, $\mathbb{D} = \{S : (S.PCell, S.\mathsf{Sig}) | S.\mathsf{Sig} \geq minsig\}$.*

If $k = |\mathcal{O}|$ and $minsig = 0$, $\mathbb{D}$ becomes equivalent to a full cube because all subspaces and all object scores would be materialized. Obviously, the storage cost would be very expensive because there could be an exponential number of subspaces as well as a large number of objects. In practice, we select $minsig > 0$, so that many subspaces being insignificant will be ignored, resulting in a small number of precomputed subspaces. Further, we have $k \ll |\mathcal{O}|$, meaning that the size of each promotion cell is also small. As a result, the promotion cube would be much more compact than the corresponding iceberg cube, which is in turn much smaller than the corresponding full cube. In fact, even for large data sets, the promotion cube is able to reside in main memory. Also, it can be efficiently computed using existing techniques like [17].

The promotion cube contributes to the online query execution by enhancing subspace pruning. It provides non-trivial upper bound and/or lower bound promotiveness scores to precomputed subspaces. More specifically, given target object t_q and target subspace S, the computation of the upper bound promotiveness $S.\overline{\mathsf{P}}$ (and the lower bound $S.\underline{\mathsf{P}}$) can be separated into the following 3 cases.

- $S \in \mathbb{D} \wedge M^S(t_q) \in S.PCell$: We can obtain the exact values of $S.\mathsf{Sig}$, $S.\mathsf{Rank}$, and $S.\mathsf{P}$ (*i.e.*, $S.\overline{\mathsf{P}} = S.\underline{\mathsf{P}} = S.\mathsf{P}$);
- $S \in \mathbb{D} \wedge M^S(t_q) \notin S.PCell$: Here exact $S.\mathsf{Sig}$ can be obtained, and Rank can be lower-bounded as $S.\underline{\mathsf{Rank}} = k + 1$. $S.\overline{\mathsf{P}}$ can be computed correspondingly;
- $S \notin \mathbb{D}$: This means that Sig can be upper-bounded as $S.\overline{\mathsf{Sig}} = minsig$. $S.\overline{\mathsf{P}}$ can be computed accordingly.

The integration of the promotion cube with the query execution algorithm PromoRank++ (Table 3) is as follows. First, when initializing the promotiveness bound for each target subspace S_i (Line 2), instead of assigning a trivial value, the upper bound $S_i.\overline{\mathsf{P}}$ and in some cases the lower bound $S_i.\underline{\mathsf{P}}$ can be computed as described above. Second, instead of initializing $\underline{\mathsf{P}}$ to be 0 (Line 6), we let it be the R-th largest lower bound promotiveness among all target subspaces (*i.e.*, the R-th largest $S_i.\underline{\mathsf{P}}$). The rest of the algorithm remains unchanged.

While in principle each promotion cell contains only the k largest aggregate scores, $M_1, M_2, \ldots, M_k$, a heuristic to further reduce space is to materialize only a subset of these scores. Given another cube parameter, cell size k', where $k' \leq k$, one can select k' aggregate scores out of the k scores at evenly spaced ranks. For example, if $k = 50$ and $k' = 5$, we materialize $M_{10}, M_{20}, \ldots, M_{50}$. For a target object, its upper and lower bound ranks are obtained by comparing its aggregate score with the k' materialized scores, and then the promotiveness bounds can be computed.

Unlike traditional data cubes which directly store aggregated results that users are interested in, the promotion cube complements the online query execution through offline preprocessing. Since the promotion cube will sit in memory, the additional query execution overhead for computing the bounds for target subspaces can be neglected. The parameters, *minsig*, k, and k', allow users to control the tradeoff between the online and offline costs so that users may select these parameters to yield the desired tradeoff. On the other hand, the parameters do not restrict the freedom of queries; that is, given $\mathbb{M}$, the promotion cube supports arbitrary promotion queries and guarantees the correctness of results.

5. AVOIDING SPURIOUS PROMOTION

In our query model, the promotiveness measure P consists of two components, the Rank measure, to make the target object prominent in a result subspace, and the Sig measure, to penalize subspaces being too sparse or too specific. There are cases, however, that the promotiveness measure fails to guarantee the meaningfulness of results.

Example 7. (SPURIOUSLY PROMOTIVE SUBSPACES) *Michael Jordan is the top scorer among all players born in February and the top scorer on sunny days.*

The example illustrates that the two subspaces {*BirthMonth* = *February*} and {*Weather* = *Sunny*}, both having high target object ranks and are neither too sparse nor too specific, are nevertheless meaningless. Clearly, there exists no causal relationship between *BirthMonth/Weather* and NBA players' ranking. Such kind of "promotive" subspaces cannot be justified and thereby having no true promotional value. We call them *spuriously promotive subspaces* and formally discuss how to avoid them in this section.

We observe that the key difference between a spuriously promotive subspace and a truly promotive one lies in that the former involves at least one *spurious dimension*. For example, *Weather* is a spurious dimension when promoting the player. Intuitively, such a spurious dimension has no correlation with object ranking. When conditioning on spurious dimension values, the object score distribution would not be significantly changed; in other words, the

Target object	Top-3 promotive subspaces	Rank	ObjCount	Top-%
Michael Jordan	{*}	3	3460	0.09%
	{Position=Guard}	1	1417	0.07%
	{Team=Chicago Bulls}	1	283	0.35%
	{Year=1984 (ties: 1986-1992, 1995-1997)}	1	380	0.26%
LeBron James	{*}	251	3460	7.3%
	{CareerStage=Young, Position=Guard}	4	1385	0.3%
	{CareerStage=Young}	14	3387	0.4%
	{Team=Cleveland Cavaliers}	1	278	0.4%
Al Jefferson	{*}	827	3460	23.9%
	{Year=2007}	11	451	2.4%
	{Position=Forward, Team=Boston Celtics}	24	139	17.3%
	{Team=Minnesota Timberwolves}	27	143	18.9%
Raymond Felton	{*}	930	3460	26.9%
	{Year=2006, CareerStage=Young}	5	142	3.5%
	{Year=2005, CareerStage=Young}	9	139	6.5%
	{Coach=Bernie Bickerstaff}	10	128	7.8%
Carlos Delfino	{*}	1337	3460	38.6%
	{Position=Guard, Team=Detroit Pistons}	33	132	25.0%
	{Coach=Flip Saunders}	32	92	34.8%
	{Team=Toronto Raptors}	36	132	27.3%

Table 4: A case study on the NBA data.

promotion of the target object would be merely due to random perturbations of ranking.

Definition 6. (SPURIOUS PROMOTION) *A subspace dimension is* spurious *when it is statistically independent of the score distribution. Any promotive subspace which contains a non-star value in some spurious dimension is considered to be* spuriously promotive.

Now, given a data set $\mathcal{D}$ (n base tuples), a subspace dimension A, and the score dimension I_{score}, our goal becomes to determine whether or not A and I_{score} have any correlation. Suppose A induces a partitioning of all base scores in I_{score} into θ (*i.e.*, A's cardinality) groups of scores, denoted as $\{s_j^1\}_{j=1}^{\varphi_1}$, $\{s_j^2\}_{j=1}^{\varphi_2}$, $\ldots$,$\{s_j^\theta\}_{j=1}^{\varphi_\theta}$, where φ_i denotes the cardinality of the i-th group ($1 \leq i \leq \theta$) and $\sum_i \varphi_i = n$. These groups are considered as samples drawn from θ underlying populations. Note that a special case is that when A induces a partitioning of all objects, we can instead partition their aggregate scores into θ groups and hence $\sum_i \varphi_i = |\mathcal{O}|$. We employ the Analysis of Variance (ANOVA) test [14] to determine if significant difference exists between these group means or they only differ by chance. The idea of the ANOVA test is to compare these sample scores' between-group sum of squared deviation (SS_B) to their within-group sum of squared deviation (SS_W). The closer they are, the higher the probability that dimension A has no effect on I_{score}. Specifically, let the null hypothesis H_0 state that the mean is the same for all groups. Let σ_i and μ_i be the sum and average score of the i-th group, respectively. Then let

$$SS_B = \sum_i \frac{\sigma_i^2}{\varphi_i} - \frac{(\sum_i \sigma_i)^2}{n},$$

$$SS_W = \sum_i \sum_j (s_j^i - \mu_i)^2.$$

The F-ratio for dimension A can be calculated:

$$F(A) = \frac{SS_B/(\theta - 1)}{SS_W/(n - \theta)}.$$

Let $F_c(A)$ be the sample scores' corresponding critical value determined by $\theta - 1$, $n - \theta$ (or $|\mathcal{O}| - \theta$ for the special case), and a given Type I error probability α (*e.g.*, 0.05). If $F(A) \geq F_c(A)$, H_0 can be rejected; otherwise, we conclude that A does not significantly influence I_{score} and thus A is a spurious attribute.

ANOVA assumes normality of score distribution and homogeneity of score variances across different groups. When the assumptions are violated, one may alternatively apply power transformations to the data or employ non-parametric methods like Kruskal-Wallis test [14].

Computational complexity: To avoid spurious promotion, we preprocess the given data set $\mathcal{D}$ by removing all spurious dimensions based on the ANOVA method and, during query execution, avoid using any spurious dimension.

This preprocessing entails one-pass over the data to compute the F-ratio for each subspace dimension. This requires $O(nd)$ time and $O(\sum_{i=1}^{d} \theta_i)$ space complexity, where d is the total number of subspace dimensions and θ_i is the cardinality of the i-th subspace dimension.

6. EXPERIMENTAL EVALUATION

In this section, our goal is to (i) verify the effectiveness of the promotion query through case study, and (ii) analyze the performance (in both time and space) of our proposed algorithms. We break down our evaluation into 3 data sets and summarize our results as follows:

NBA data set [2] (Section 6.2) Conduct case study to show how NBA players can be effectively promoted. Because this data set is small, no performance result will be reported.

DBLP data set [1] (Section 6.3) Conduct both case and performance studies. We confirm that the search results match our intuition well, and the proposed algorithms perform significantly better than baseline algorithms.

TPC-H [3] (Section 6.4) By generating synthetic data sets using a wide range of parameters, we show that the proposed algorithms consistently outperform the baseline ones.

6.1 Implementation

All experiments were done on a machine with a Pentium 3GHz processor, 2GB of memory, and 160G hard disk. We implemented the PromoRank, PromoRank++, and PromoCube algorithms. Our performance evaluation is based on two measures: *query execution time* and *space overhead*. PromoRank is considered a baseline for query execution time, while to evaluate the space overhead of PromoCube, we compare it to a traditional iceberg cube which,

Target object	Top-3 subspaces by P_1	Rank	ObjCount	Top-%	Top-3 subspaces by P_2	Rank	ObjCount	Top-%
David DeWitt	{*}	376	451316	0.08%				
	{Database}	16	65321	0.02%	{PDIS}	1	318	0.31%
	{1990}	2	13170	0.02%	{Database, SIGMOD}	1	1784	0.06%
	{SIGMOD}	2	3519	0.06%	{Database, 1985}	1	556	0.18%
Yufei Tao	{*}	3325	451316	0.74%				
	{Database, 2003}	11	6707	0.16%	{ICDE, 2004}	1	334	0.30%
	{Database, 2004}	18	8877	0.20%	{Data Mining, Info. Retrieval, 2004}	2	690	0.29%
	{ICDE}	30	4822	0.62%	{SIGMOD, 2008}	5	471	1.06%

Table 5: Promotion query results on the DBLP data using different promotiveness measures.

for each subspace passing some given *minsig* threshold, fully materializes all object aggregate scores.

The source code was written in C# and compiled using Microsoft Visual C# 2008 in Windows XP. All query processing algorithms were executed in the main memory without any disk access (PromoCube resides in memory).

6.2 The NBA Data Set

Data characteristics: The data set contains 18050 base tuples, each recording a player's statistics in a particular year. Each base tuple contains 5 subspace dimensions, namely *Year* (62 values), *CareerStage* (2 values, "*Young*" or otherwise), *Position* (3), *Team* (68), and *Coach* (220). We used *PlayerID* (3460) as object dimension, and used statistics dimensions such as *Points*, *Rebounds* as score dimensions.

Promotiveness measure: *SUM* was chosen as the measure $\mathbb{M}$ to aggregate over *Points* in order to generate ranking. The promotiveness measure was set to $\mathsf{P}_1 = -\mathsf{Rank} - 2^{-\lceil \log(\mathsf{TupleCount}/n) \rceil}$, where $2^{-\lceil \log(\mathsf{TupleCount}/n) \rceil}$ is a penalty equivalent to 2^l (l being a non-negative integer) when the subspace selectivity $\frac{\mathsf{TupleCount}}{n}$ falls in range $(\frac{1}{2^{l+1}}, \frac{1}{2^l}]$.

We first conduct a case study. Table 4 shows the top-3 promotive subspaces for 5 representative players. For comparison, it shows their global and local ranks, as well as their precise ranks in percentage. We find these results to match the reality very well. For example, the case for Michael Jordan has been used in the Example 2 in Section 1. For LeBron James, he is ranked only 251st among all players; however, our results reveal more exciting facts such as that James being a talented young player (*i.e.*, 14th out of 3387 young players). For other globally low-ranked players, their promotive subspaces make sense as well.

The above results are obtained using PromoRank++, where the object pruning method estimates the TupleCount of a subspace using the product of the selectivities of the subspace' dimension values. In fact, the results in Table 4 are accurate, because the promotiveness measure being used is not sensitive to selectivity.

Handling spurious promotion: Figure 8 shows the effectiveness of the ANOVA test for detecting spurious dimensions. In addition to the 5 subspace dimensions mentioned earlier, another 2 dimensions *BirthMonth* (12) and *RandomDim* (100) were also considered here for comparison. *BirthMonth* records a player's birth month and was extracted from [2], and *RandomDim* is a uniformly randomly generated dimension. The score dimension was set to *Rebounds*. Figure 8 displays the F-ratio for each of the 7 dimensions (on log scale/in a decreasing order) and the corresponding critical values at $\alpha = 0.05$. The null hypothesis for *BirthMonth* and *RandomDim* cannot be rejected since their F-ratios are smaller than the critical values. This means that they have no significant correlation with the score dimension and thus the subspace object ranking. On the other hand, the remaining 5 subspace dimensions have F-ratios significantly larger than the critical values, exhibiting that they are strongly correlated with player rankings. These results have empirically verified that spurious dimensions indeed can be separated from meaningful subspace dimensions by the statistical test.

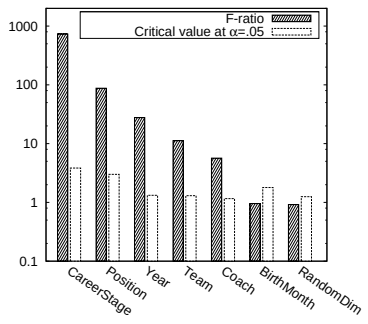


Figure 8: Correlation test on NBA dimensions.

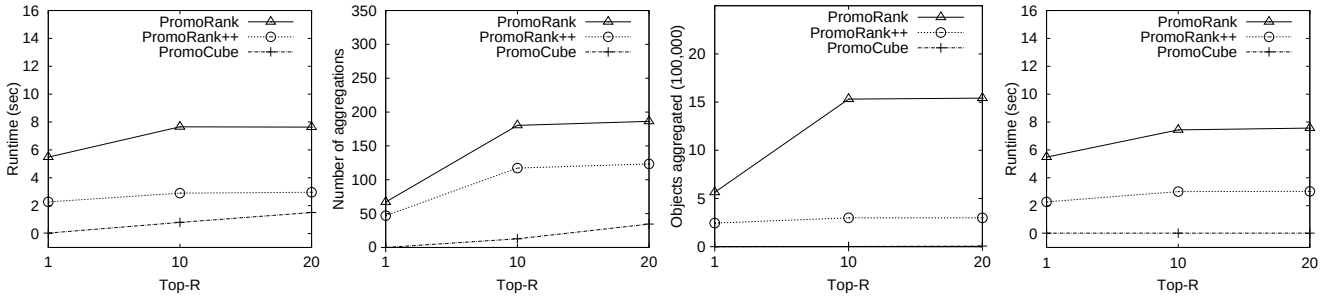
6.3 The DBLP Data Set

Data characteristics: We extracted 1,763,888 base tuples from the DBLP data set, each in the format (*Author, Conference, Year, Title*). The cardinalities of *Author*, *Conference*, and *Year* are 451316, 2506, and 50, respectively. We used *Author* as object dimension and consider *Conference* and *Year* as subspace dimensions. We also extended *Title* to 4 boolean subspace dimensions, "*Database*", "*Data Mining*", "*Information Retrieval*", and "*Machine Learning*" using a manually constructed keyword-to-field mapping, so totally there are 6 subspace dimensions. For example, a base tuple with title containing "*bayesian*" would have its *"Machine Learning"* dimension set to *"true"*. The selectivities of the *"true"* value of the 4 boolean dimensions are between $72K$ and $226K$. We used *COUNT* (*i.e.*, number of publications) to compute aggregate scores.

Promotiveness measure: Often users may want to enforce significance constraint on subspaces (much like an iceberg condition) rather than imposing penalty on selective subspaces. Therefore, in addition to P_1, we experimented with another measure P_2. Specifically, let P_2 be $\mathsf{Rank}^{-1} \cdot \mathcal{I}\{\mathsf{TupleCount} \geq 100\}$; that is, only subspaces having ≥ 100 base tuples (*i.e.*, a community with > 100 papers) will be considered, whereas the remaining ones will be pruned. Intuitively, this measure prevents us from searching too "deeply" in the target lattice.

Table 5 shows the top-3 query results for two typical authors using P_1 and P_2. Not surprisingly, the subspaces characterize the authors' strengths well. We can see that P_1 prefers subspaces with large population and reasonably high ranks, whereas P_2 prefers absolute high ranks as long as the subspace meets the significance constraint. It is worth mentioning that there is no universally best promotiveness measure, and it is up to the user to choose the proper measure for her task.

Performance: We created a workload consisting of a set of 10 randomly selected authors with ≥ 10 papers as target objects. For PromoCube, we chose *minsig* $= 100$; it turns out that $k = k' =$



(a) Runtime vs. top-R. (b) Number of subspace aggregations vs. top-R. (c) Number of objects aggregated vs. top-R. (d) Runtime on promotiveness measure P_1.

Figure 9: Performance results on the DBLP data.

10 would suffice to materialize almost all distinct aggregate scores in significant subspaces since paper counts are often small integers. Thus, the resulting space overhead for the data set, 310*KB*, can be regarded trivial, and we do not compare it with iceberg cube.

We report in Figure 9(a) the query execution time of the PromoRank, PromoRank++, and PromoCube algorithms when varying R, the number of subspaces to be returned. The promotiveness measure was set to P_2. We can see that all these methods become slower when R increases, as expected. PromoRank++ is consistently 2 to 2.5 times faster than PromoRank, showing the superiority of the proposed pruning techniques. PromoCube outperforms PromoRank by a ratio of 180, 9.6, and 5 when R is set to 1, 10, and 20, respectively. In particular, PromoCube performs extremely well when R is small, because in such cases, the promotion cube can directly return the result using $O(1)$ lookup time and terminate early; when R increases, additional online cost is incurred to aggregate non-pruned search space.

To explain the differences in query execution time, in Figure 9(b), we plot the total number of subspaces aggregated by each of the 3 algorithms with respect to R. Clearly, this figure shows that the query execution time is linearly correlated with the number of subspace aggregated. Moreover, Figure 9(c) shows the total number of objects aggregated by each algorithm with respect to R. Compared to PromoRank, the baseline strategy, PromoRank++ dramatically reduces the number of objects aggregated, due to the power-law distribution of author aggregate scores. In fact, PromoRank++ is able to prune out the long tail of over 50% authors with small paper count at the very initial recursion, resulting in significant cost saving of subsequent object aggregation and data partitioning.

Figure 9(d) displays the runtime comparison of the 3 algorithms based on the promotiveness measure P_1, which enforces penalty on subspaces. Again, PromoRank++ is up to 2.5 times faster than PromoRank, verifying the effectiveness of the pruning techniques. The total number of subspaces and objects aggregated are quite similar to previous results so we do not plot them here. Moreover, PromoCube consistently performs extremely well due to the relatively small domain of aggregate scores. Actually, for all the queries being tested in here, looking up the promotion cube suffices to answer them. In other words, the target object's ranks were already precomputed in the top subspaces.

6.4 The TPCH Data

Now we compare the performance of algorithms on the synthetic TPCH data [3].

Default data characteristics: We generated a default data set with scale factor set to 1 and extracted the *lineitem* file containing 6,001,215 base tuples and 16 dimensions. We used the following 6 dimensions as subspace dimensions: *l_shipdate* (2526), *l_quantity* (50), *l_discount* (11), *l_tax* (9), *l_linenumber* (7), and *l_returnflag* (3). Also, we used *l_suppkey* (10,000) as object dimension and *l_extendedprice* (ranges from 901.00 to 104949.50) as score dimension. Let *SUM* be the aggregate measure and objects are ranked in its descending order.

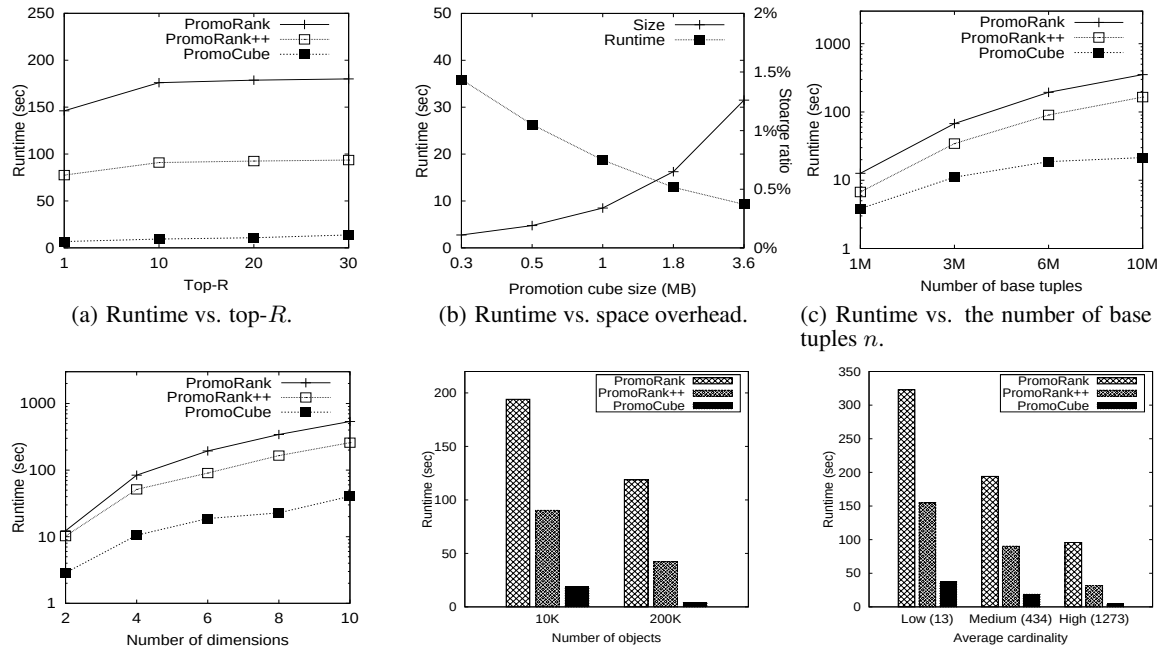
Performance: In this subsection we consider P_1 as well as a promotiveness measure P_3 with iceberg constraint TupleCount $\geq$ 1000. For PromoCube, we set *minsig* $= 1000$; for each subspace, we considered the largest $k = 1000$ aggregate scores and materialized $k' = 8$ of them at evenly spaced ranks. The resulting space overhead is 978.3*KB*, in contrast to 277.5*MB*, the size of the iceberg cube with condition TupleCount ≥ 1000 (*i.e.*, for each subspace passing the condition, materialize all aggregate scores). All results reported here were averaged over a set of 5 randomly generated target objects.

Figure 10(a) compares the performance of PromoRank, PromoRank++, and PromoCube in terms of R (using P_1). PromoRank++ is approximately 2 times faster than PromoRank, while PromoCube is about 20 times faster than PromoRank. Compared with DBLP, the synthetic data used here is more than 3 times larger, so more query execution time is consumed by all methods. However, even when $R = 30$, PromoCube needs only $13.7s$, showing the feasibility of promotion analysis over large data sets.

Figure 10(b) displays the relation between promotion cube size and query execution time when fixing $R = 10$ (using P_3). To generate promotion cubes with different sizes, we fix the *minsig* parameter to 1000 and k to 1000 but vary k'. The sizes of the resulting five cubes range from 0.3*MB* to 3.6*MB*, which are 0.11% to 1.26% of the corresponding iceberg cube's size, 277.5*MB*. We can see that an exponentially decreasing cube size leads to only linearly increasing query execution time. In particular, when the cube size is only 0.3*MB* (0.11% of the iceberg cube size), the runtime is $36s$, about 4.8 times faster than the baseline ($176s$ as shown in Figure 10(a)). Our results verify the robustness of the promotion cube in the sense that the query target objects' promotive subspaces may or may not be precomputed.

Now we fix the query parameters (*i.e.*, the promotiveness measure, the set of 5 target objects, and $R = 10$), and further generate different synthetic data sets by varying each of the following parameters: (i) the number of base tuples, (ii) the number of subspace dimensions, (iii) the number of objects, and (iv) the average cardinality of subspace dimensions. We report the performance results of the algorithms as follows.

First, we evaluate the algorithms on four data sets with $1M$, $3M$, $6M$, and $10M$ base tuples. We set the space overhead of Pro-



(a) Runtime vs. top-R.

(b) Runtime vs. space overhead.

(c) Runtime vs. the number of base tuples n.

(d) Runtime vs. the number of dimensions d.

(e) Runtime vs. the number of objects.

(f) Runtime vs. average cardinality.

Figure 10: Performance results on the TPCH data.

moCube to be proportional to the number of base tuples. The runtime of PromoRank, PromoRank++, and PromoCube with respect to the base tuple number n is depicted in Figure 10(c). As expected, PromoRank++ is about 2 times faster than PromoRank, because both partitioning and aggregation costs are linear to n. On the other hand, PromoCube is increasingly faster with respect to n, because the PromoCube prunes subspaces before any online aggregation happens, which is unlike PromoRank++ that prunes subspaces during the online aggregation process. Therefore, the actual aggregation and partitioning cost saving of PromoCube is much larger than that of PromoRank++.

Second, we vary the number of subspace dimensions, d, for the default data set with $6M$ base tuples. In addition to the 6 subspace dimensions mentioned earlier, 4 more dimensions were included for this test case: *l_commitdate* (2466), *l_linestatus* (9), *l_shipmode* (7), and *l_shipinstruct* (4). As shown in Figure 10(d), the gap between PromoRank++ and PromoRank is not large when $d \leq 4$. This is because the total number of target subspaces itself is quite small, so the pruning techniques would not be effective. When $d \geq 6$, the number of target subspaces becomes larger, so more interdependent relationships can be exploited. Our conclusion is that increasing the number of subspace dimensions may not result in an increasing gain of PromoRank++ over PromoRank; nevertheless, it does make PromoCube more efficient (relatively) because more likely there will be some subspaces where a target object ranks high.

Third, to test the relation between the total number of objects $|\mathcal{O}|$ and query execution time, we replace *l_suppkey* (10,000) with *l_partkey* (200,000) as the object dimension on the default data set. As shown in Figure 10(e), when $|\mathcal{O}| = 200K$, all algorithms' absolute execution time is slower than when $|\mathcal{O}| = 10K$, because the more objects, the less the number of target subspaces there will be (*i.e.*, each object will appear in less base tuples). On the other hand, the speedup ratio of PromoRank++ goes from 2 for $10K$ objects to 2.8 for $200K$ objects. Intuitively, more objects tend to make the subspace aggregation cost outweigh the partitioning cost relatively, which would magnify the effectiveness of subspace pruning, which aims to reduce the number of subspace aggregations. For the same reason, PromoCube goes from 10 times faster for $10K$ objects to 30 times for $200K$ objects.

Lastly, we vary average cardinality by selecting 3 different sets of 6 subspace dimensions from the default $6M$ data set. We label these sets as "low" (average cardinality 13), "medium" (*i.e.*, the default set of 6 subspace dimensions studied earlier with average cardinality 434), and "high" (average cardinality 1273). As displayed in Figure 10(f), PromoRank++ and PromoCube are 3 and 20 times faster than PromoRank on the high-cardinality data set, and 2.1 and 8.7 times faster on the low-cardinality data set. It is found that both algorithms favor large cardinalities, where aggregates become very sparse. In such cases, the aggregate scores would be equal (*i.e.*, to base scores) across parent-child subspaces, thereby providing a tighter lower bound for Rank.

7. DISCUSSION

7.1 Related Work

Data mining for marketing: Promotion is one of the 4 P's in marketing: Product, Price, Place, and Promotion [13], which serves for developing brand, building awareness, *etc.*. The idea of leveraging ranked results for promotion is ubiquitous. In online search-based advertising, researchers have proposed methods for optimizing the effectiveness of promotion (*e.g.*, in terms of click-through rate) given a keyword bidding budget [6]. There are other data mining studies for marketing, including [12] for microeconomic studies, [16] for maximizing product retrieval frequency, and [15] for dominant relationship analysis. Our problem takes a unique

perspective in that we explicitly exploit ranking for promotion by comparing objects in subspaces.

Ranking queries: Ranking has been extensively studied in Web search, databases, and other fields. Ranking (top-k) query model augments the traditional boolean query model by enabling ad-hoc search and retrieval. Numerous techniques are developed for effective and efficient ranking queries [9, 11, 7]. Notably, ranking optimization is discussed for multidimensional data and aggregate queries [19, 4]. Our query model can be distinguished from them as the target object is specified upfront as user *input*, as opposed to be *output*. Another difference is that we use rank as a measure for promotion, while previous ranking is for returning a digestible set of results to user. Toward the problem of finding top-k attributes, [8] investigates selecting the most useful attributes for explaining ranked tuples, and [16] aims to find the best attributes to maximize for a given workload the number of queries which can retrieve a tuple. Their objectives are not to explore ranking for promotional purposes and they focus on the attribute-level.

OLAP and decision support: Technically, promotion queries are similar to OLAP and decision support queries as far as multidimensional analysis is concerned. The data cube model has been well-studied to enhance explorative analysis [10, 15]. The iceberg cube model [5] is proposed to answer iceberg queries efficiently. These techniques rely on monotonicity and/or convexity of an aggregate measure to speedup query processing and/or reduce storage size. The promotiveness measure, however, does not have monotonicity, anti-monotonicity, or convexity property, and thus previous techniques are unable to handle promotion analysis. Other multidimensional anomaly and outlier detection models cannot replace promotion analysis since they do not deal with ranking.

7.2 Extension

There could be various ways to model promotion applications and our work addresses an interesting and technically difficult one. Here we discuss several directions for future study.

Group promotion: Instead of promoting a single target object, users may want to promote a target group of objects. For example, the target group could be a collection of items in a product package. For such group promotion, the promotiveness can be measured by the average increase in rank for each object in the target group or by some other aggregate measure. The PromoRank algorithm would work with minor extensions so that the promotiveness of subspaces can be iteratively computed and the most promotive subspaces are maintained in a priority queue during query execution. How to extend the pruning techniques, on the other hand, would be an interesting open question.

Mining measure space: Mining promotive regions in the measure space is an orthogonal yet important problem. Unlike the multidimensional space that is organized in a finite number of cells, the measure space is often numerical and thus impossible to be exhaustively enumerated. On the other hand, it would be very useful for decision makers to understand the product attributes and position products. For example, knowing by what criteria a product is successful (*e.g.*, by sales or by customer rating) can help further position and promote it.

Promotion in social networks: A social network, with each node and link carrying some multidimensional information, may need promotion analysis as well to promote objects in such a network or in its surrounding subnetworks. However, the promotion measures of an object could be related to certain network property, such as network density, connectivity, and centrality; the computation of such a promotion measure could be closely related to the network topological structure and the node/link values. The methods for promotion analysis will need to be re-examined in such networks. We propose to perform network-based precomputation so that an initial evaluation of the nodes and links can be done before query time such that the online query-based promotion computation can avoid searching many hopeless paths. The detailed search strategy is left as an interesting theme for future research.

8. CONCLUSION

We have introduced the promotion analysis problem. New pruning techniques as well as the promotion cube approach have been proposed for efficiently answering the promotion query. Our comprehensive experiments verified that the promotion query is able to discover meaningful results, and the efficiency of our proposed algorithms significantly outperform the baseline solutions. We believe that this work opens up new horizons for research and we plan to investigate them in depth in the future.

Acknowledgements: We thank the anonymous reviewers for their helpful comments. The work was supported in part by the U.S. National Science Foundation grants IIS-08-42769 and BDI-05-15813, and the Air Force Office of Scientific Research MURI award FA9550-08-1-0265.

9. REFERENCES

[1] DBLP. http://www.informatik.uni-trier.de/∼ley/db/.
[2] NBA data set, http://www.basketballreference.com.
[3] TPC-H. http://www.tpc.org/tpch/.
[4] N. Bansal, S. Guha, and N. Koudas. Ad-hoc aggregations of ranked lists in the presence of hierarchies. In *SIGMOD*, pages 67–78, 2008.
[5] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, pages 359–370, 1999.
[6] C. Borgs, J. T. Chayes, N. Immorlica, K. Jain, O. Etesami, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *WWW*, pages 531–540, 2007.
[7] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.
[8] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD*, pages 395–406, 2006.
[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.
[11] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
[12] J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Min. Knowl. Discov.*, 2(4):311–324, 1998.
[13] P. Kotler and K. Keller. *Marketing Management*. Prentice Hall, March 2008.
[14] R. Kuehl. *Design of Experiments: Statistical Principles of Research Design and Analysis*. Duxbury Press, 2000.
[15] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD*, pages 659–670, 2006.
[16] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*, pages 356–365, 2008.
[17] Z. Shao, J. Han, and D. Xin. Mm-cubing: Computing iceberg cubes by factorizing the lattice space. In *SSDBM*, pages 213–222, 2004.
[18] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 193–204, 1999.
[19] T. Wu, D. Xin, and J. Han. Arcube: supporting ranking aggregate queries in partially materialized data cubes. In *SIGMOD Conference*, pages 79–92, 2008.