Maximizing a Record's Standing in a Relation

Abstract—Given a database table with records that can be ranked, an interesting problem is to identify selection conditions for the table, which are qualified by an input record and render its ranking as high as possible among the qualifying tuples. In this paper, we study this standing maximization problem, which finds application in object promotion and characterization. After showing the hardness of the problem, we propose greedy methods, which are experimentally shown to achieve high accuracy compared to exhaustive enumeration, while scaling very well to the problem input size. Our contributions include a linear-time algorithm for determining the optimal selection range for an ordinal attribute and techniques for choosing and prioritizing the most promising selection predicates to apply. Experiments on real datasets confirm the effectiveness and efficiency of our techniques.

Index Terms—Standing Maximization Problem, NP-Hardness, Relational Databases.

1 Introduction

Certain classes of database operations, like top-k [6] and skyline queries [2], rank the records in a relation according to their values in some attributes and/or user preference functions. With the help of such queries, superior objects can be identified. However, there may be objects, which stand-out not among *all* their peers, but only among records which qualify certain selection conditions. Finding these conditions helps identifying the criteria that make a given object important and facilitate its characterization and promotion.

TABLE 1
A relation with CS PhD graduates

name	age	location	expertise	publications
Brown	30	N. America	systems	14
Smith	27	N. America	databases	8
Suzuki	32	Asia	theory	9
Müller	28	Europe	theory	15
Dubois	26	Europe	systems	12
Martin	31	Europe	databases	17
Kim	28	Asia	databases	10
Chen	26	Asia	theory	12
Gupta	26	Asia	systems	13

As an example, consider the relation shown in Table 1, which stores information about CS PhD graduates. Assuming that the last attribute (publications) is used as a measure for the quality of the graduates, Kim does not have a good ranking. However, if we restrict the relation to include just Kim and Smith, then Kim ranks 1st in this set; the restriction can be imposed by the selection conditions ($\texttt{age} < 30$) and ($\texttt{expertise}$ = 'databases').

Problem Definition. Motivated by this observation, in this paper we study the *standing maximization problem* (SMP), which takes as input a relation $R(\mathcal{D}; M)$, a *query* tuple $t_q \in R$, and a *support* threshold sup, $0 < sup \leq 1$. $R.\mathcal{D}$ is a set of *predicate* attributes and $R.M$ is used as the *measure* attribute for ranking the tuples in R. Without loss of generality, we assume that if $t.M > t'.M$, for two tuples $t, t' \in R$, then t is considered better than t'. The objective is to find a *conjunction* of selection predicates C on $R.\mathcal{D}$, such that (i) t_q is included in $\sigma_C R$, (ii) there are at least $sup \cdot |R|$ tuples in $\sigma_C R$, and (iii) the *percentile rank* $pr(t_q, \sigma_C R)$ is maximized. $pr(t_q, \sigma_C R)$ is the percentage of tuples in set $\sigma_C R$ whose values on $R.M$ are smaller than or equal to $t_q.M$.

SMP basically strives to identify a set of selection predicates that maximize the input tuple's t_q ranking among those that qualify the predicates. In the example of Table 1, t_q = Kim could be an individual who, using our query, wants to promote herself in job applications using $\texttt{publications}$ as a measure. By identifying selection predicates that make her stand-out among all graduates that qualify the predicates, Kim can advertise herself using the most appropriate features (e.g., she is the best among people in her age group and expertise). The support threshold sup guarantees that there are enough tuples in the selection result for the ranking to have statistical significance. An appropriate value for sup can be chosen using similar methods as for the mining association rules problem. In addition, we require that there is a single predicate for each attribute; a disjunction of predicates can include tuples which are inferior to t_q, but together with t_q they may not form a natural group. For example, the predicate-set ($\texttt{age} > 26$), ($\texttt{location}$ = 'Asia' or 'N. America') and ($\texttt{expertise}$ = 'databases' or 'theory') returns tuples {Smith, Suzuki, Kim}, in which Kim prevails, however, the predicates do not define a natural group of tuples with a concise description. SMP and the proposed techniques in this paper are not restricted to using a given ranking attribute $R.M$; any arbitrary criteria could be used for ranking the records.

Applications. Object promotion is the main application of the SMP query; still, there are additional uses of an SMP query result. First, the result can be used to *characterize* the input object among its peers, by finding out what is special about it. Second, by identifying the selection conditions that make the object stand-out, we also identify its competitors (i.e., the peers that also satisfy these conditions) and we are able to assess the effort required to improve its rank (e.g., how many additional

publications Kim would need to become first in her age group). The two examples that follow illustrate the real-life applicability of SMP.

Example 1 (University Ranking): A PR officer of the Hong Kong University of Science and Technology (HKUST) would like to promote HKUST as a prominent university. By checking a world university ranking (www.topuniversities.com) he realizes that HKUST only ranks 33rd among all universities. However, further analysis reveals that HKUST is the top university among all universities that are aged under 50 (i.e., universities established within (1963, 2014]).

Example 2 (E-commerce): A car owner wants to sell his car via a second-hand auto trading website, but the price he sets is quite high, which does not make the car attractive to potential buyers. However, with the help of an SMP query, he discovers that his car is actually the second cheapest one among those cars that are made by BMW and they have mileage less than 10,000 miles. The car owner now has more specific information to put in his ad in order to promote his car; he may even make a minor adjustment to the price in order to make his car the cheapest in its group.

Contributions. We show that SMP is a hard problem, due to the exponential number of predicate combinations that should be considered. In addition, in this paper we focus on solving SMP for the case where the selection attributes are ordinal, which renders the space of possible predicates for a given attribute quadratic to the attribute's domain size. In view of this, we propose greedy methods which explore the search space only partially, striving to identify a sub-optimal SMP solution of high quality. Our first method, BA, greedily picks the most promising attribute and the best selection predicate on it at a time; while this method is extremely fast, it explores a very limited part of the search space. Our second method, DBA, extends BA to consider a small number of predicates for the (greedily) best attribute at a time. Finally, EDBA considers all possible attribute orderings and a small number of predicates for each of them. Our contributions include a linear-time algorithm for determining the optimal selection range for an ordinal attribute (used by all our methods) and techniques for choosing and prioritizing the most promising selection predicates to apply (in DBA and EDBA). As we demonstrate experimentally, EDBA finds a solution with quality close to the optimal, while being orders of magnitude faster than the baseline solution. BA and DBA trade time with quality; they are faster than EDBA but their results are not as good. EDBA can also be applied as a *progressive* algorithm which, if given more time, improves its result, and can be stopped as soon as the user is happy with the solution found so far.

Outline. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 outlines baseline solutions for SMP, which compute the problem exactly but are too expensive to be practical, and presents our proposed methods for finding an approximate solution to SMP fast. Section 4 includes an experimental evaluation and Section 5 concludes the paper.

2 Related Work

Rank-based Analysis and Query by Output. Das et al. [4] study the problem of identifying a small set of attributes in a relation which are the most influential in the ranking of the results of a given query. Our problem is different, since we are trying to define selection conditions that maximize a given record's standing. Tran et al. [11] investigate the problem of finding an SQL statement that produces results which include a given set T of input tuples. This problem is relevant to ours; however, in our case T is not predefined, and it could be any query result that includes our input object and other objects that are (mostly) inferior to it. Miah et al. [8] aim at finding a set of attributes that maximize the retrieval hit of a target tuple given a query workload (again, a different problem to SMP).

Promotion Analysis. The most related previous work to ours [15], [16] studies a generalization of SMP: for each object there could be multiple tuples in the relation and after applying the selection condition, the tuples that correspond to each object are *aggregated* to a single score based on which the objects are ranked. In contrast, in our setting, there is a single value for each object in the measure attribute and selection conditions do not affect the measure and the ranking. Wu et al. [16] solve this generalized SMP for the case where all attributes are binary or categorical. Given the high cost of exploring the possible selection conditions, they resort to a materialization approach. For each subspace (i.e., combination of values for a subset of attributes), a summary of the objects' ranking score distribution is precomputed and materialized. Given a query object, every subspace is examined and the materialized ranking information is used to derive an lower/upper bound of the query object's ranking in the subspace. For subspaces which cannot be pruned, the exact ranking of the query object is computed and it is determined whether the subspace can be returned as a result or not. In [15], this framework is extended to also consider ordinal attributes. Although the materialization and exploration approaches of [16] and [15] can directly be used to solve SMP, they have very high storage requirements and their cost is extremely high, as we demonstrate in Section 4. A variant of the problem, where only a fixed number of subspaces is considered instead of the whole set of subspaces, is studied in [17]: an approximate solution to the original problem is derived (considering binary attributes only). All solutions above explore a large number of subspaces, thus they have a high cost. Finally, a similar (but different) problem to SMP is studied in [10]: given an ever-growing append-only table, the problem is to incrementally answer, for each new tuple entering the table, the subspaces where this new tuple belongs to

and becomes a skyline object, in the context of multiple measure attributes. The basic idea is to maintain the set of skyline tuples by far and compare the new tuple with each skyline tuple, while in SMP our goal is to identify the subspaces where the query tuple t is ranked as high as possible, provided that the size of the subspace satisfies the support threshold. Moreover, the selection predicates considered in [10] are all categorical, while in our work we focus on ordinal selection predicates.

Query Refinement. Given a set of user-specified outliers in an aggregate query result, Wu and Madden [14] study the problem of finding predicates that cause the presence of these outliers; the motivation is to remove the tuples generated by these predicates and thus eliminate the outliers. The predicates investigated in [14] only apply to the tuples that compute the selected outlier results, while in our work the predicates affect the whole input. Furthermore, different to our problem settings, [14] focus on aggregate queries. Mishra and Koudas [9] investigate how to refine the predicates of a query, in order for the query results to satisfy certain user-specified cardinality constraints. This can be done by estimating the selectivity of predicates. The only constraint considered in [9] is the query output size, while in SMP, we also aim at maximizing the percentile rank of a given record.

Reverse top-k queries. A reverse top-k query [12] finds the weights to be given to attributes of a relation in a top-k (i.e., preference) query, in order for a given object to be in the top-k result. Based on the reverse top-k definition, Vlachou et al. [13] also define and solve the problem of finding the top-m most influential products (which appear in the top-k sets of most preference queries). Arvanitis et al. [1] investigate the similar problem of finding the set of customers that consider a given product attractive, based on their preferences. Das et al. [5] study how to automatically provide a meaningful interpretation of the average rating of a given item (e.g., a movie in IMDB), by identifying the features of the users that rated the item (e.g., males under 30 gave consistently high ratings). Lappas et al. [7] introduce the concept of competitiveness between products based on the relationships between the sets of their potential customers, and propose an algorithm for finding the top-k competitors of a given item. All works above study different problems; their techniques cannot solve SMP.

3 METHODOLOGY

Before presenting our solutions to SMP, we elaborate on the problem's input. The *predicate attributes* in $R.\mathcal{D}$ can be of three types: (i) ordinal, (ii) hierarchical, (iii) and binary/categorical. If an attribute is of ordinal type (e.g., `age`), we can define an equality (e.g., `age` = 28) or a range predicate on it (e.g., $26 \leq$ `age` ≤ 28). For hierarchical attributes, we assume that there is a hierarchy of values (e.g., derived by joining R with another table). For example, the `location` attribute could be hierarchical, since locations can be generalized. Thus, R may store locations at the lowest granularity (e.g., city), which could be generalized (e.g., to county, state, country, continent). For such attributes, an equality predicate can be applied at any granularity of the hierarchy which includes the value of t_q (e.g., `location` = Boston, `location` = USA). Finally, for binary and categorical attributes the only possible predicate is equality on the value of t_q. Going back to our introductory example, a solution to our problem can include an equality or range predicate on `age` and an equality predicate on `location` and `expertise`. If `location` was hierarchical, the equality predicate could be applied at any granularity.

3.1 Problem Complexity

To solve SMP we have to consider an exponential number of predicate combinations. Assume that R has m_o ordinal attributes, m_h hierarchical attributes, and m_b binary/categorical ones. The number of possible predicate combinations C to be considered is $O(\Pi_{i=1}^{m_o}(\min\{|R|, |D_i|\})^2 \cdot \Pi_{i=1}^{m_h} h_i \cdot 2^{m_b})$, where $|D_i|$ is the number of discrete values in the domain D_i of the i-th ordinal attribute and h_i is the height of the generalization hierarchy of the i-th hierarchical attribute. For each ordinal attribute there are $O((\min\{|R|, |D_i|\})^2)$ possible range predicates that contain the value of t_q; for each hierarchical attribute there are h_i possible predicates; and for each binary/categorical attribute we can choose to a predicate or not. Any conjunction of predicates can be the solution of SMP; in other words, in the general case, the percentile rank of t_q can arbitrarily change after changing the predicates to include more or fewer tuples. Formally:

Theorem 1: SMP is NP-hard.

Proof: (Sketch): We will show the hardness of SMP for the case where all attributes are boolean; problem instances with non-boolean attributes are even harder. The decision version of SMP, denoted as SMP′, is as follows: "for a given relation R and a query tuple t_q, is there a set of selection predicates C, such that $pr(t_q, \sigma_C R) \geq \lambda$, subject to $t_q \in \sigma_C R$ and $|\sigma_C R| \geq sup \cdot |R|$?". Obviously SMP′ is in NP. To prove its NP-hardness, we reduce the Set Cover Problem (SCP), which is NP-complete, to SMP′. Given a collection $\mathbb{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of a base set $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$ and an integer k, the objective of SCP is to find if there exists a subcollection $\mathbb{S}' \subseteq \mathbb{S}$ whose union is $\mathbb{X}$, subject to $|\mathbb{S}'| \leq k$. We transform this to an SMP′ instance having $\lambda = 100\%$ and construct R as follows. R consists of m predicate attributes and one measure attribute M. Each subset S_j corresponds to a predicate attribute A_j, i.e., a subset S_j is chosen if and only if C has a predicate $A_j = 1$. R contains $1 + n + m$ tuples. The predicate attribute values for the first tuple t_0 are all 1 and $t_0.M = 0$. t_0 is followed by n tuples where each tuple $t_i (1 \leq i \leq n)$ corresponds to a variable $x_i \in \mathbb{X}$. We set $t_i.A_j$ to 0 if $x_i \in S_j$ $(1 \leq i \leq n)$, and 1 otherwise. The measure attribute values for these

TABLE 2
A relation with records of cars

Car	MPG	Year	HP	Rating
o_1	20	2003	150	89
o_2	25	2011	132	85
o_3	30	2008	135	78
o_4	27	1998	150	71
o_5	22	2006	132	67
o_6	34	2007	126	64
o_7	25	2010	150	61
o_8	40	2000	138	57
o_9	25	2003	132	56
o_{10}	18	2011	138	51

tuples are 1. Finally, R contains m more tuples where $t_i.A_j = 0$ if $i - n = j$ $(n + 1 \leq i \leq n + m)$, and 1 otherwise. The measure attribute values for these tuples are set to 0. We complete building the instance by setting $sup = (m - k + 1)/(1 + n + m)$ and $t_q = t_0$. It is not hard to verify that the SCP instance has a solution if and only if there exists a set of selection predicates C such that $\sigma_C R \geq sup \cdot |R|$ and $pr(t_q, \sigma_C R) \geq 100\%$. □

For the rest of the discussion, we assume that only ordinal attributes exist, because they offer higher flexibility in imposing predicates; the other attribute types can trivially be considered as special cases of ordinal attributes. A running example demonstrating our solutions to SMP is given in Table 2 (a set of used cars on sale). The table has three ordinal predicate attributes, i.e., `MPG` (miles per gallon), `Year` (release year) and `HP` (horse power), and one measure attribute: `Rating`. Assume that the support threshold sup is 0.3, and the query object t_q is o_6, whose global percentile rank is only 50%. By running SMP on this example, we can find that the best set of selection predicates for o_6 are {`MPG`: (22, 40], `Year`: (1998, 2007]}, which renders o_6 rank the highest (i.e., 100% percentile rank) among $\{o_6, o_8, o_9\}$.

3.2 Baseline Methods

Algorithm 1 Naive Algorithm

```
1: G := R; Preds = ∅; bestrank := qual(G);
2: bestG := G; bestPreds := ∅;
3: procedure NAIVERANGE(G, Preds)
4:     if all attributes are in Preds then
5:         if qual(G) > bestrank then
6:             bestrank := qual(G);
7:             bestG := G; bestPreds := Preds;
8:     else
9:         Pick any attribute A not in Preds
10:        A.preds:=all possible predicates on A for G
11:                such that |σ_pred G| ≥ sup · |R|;
12:        for each pred ∈ A.preds do
13:            G' := σ_pred G;
14:            Preds' := Preds ∪ {pred};
15:            NAIVERANGE(G', Preds');
16: return {bestG, bestPreds};
```

A straightforward approach for solving SMP is to enumerate in a depth-first manner all possible *subspaces* (i.e., conjunctions of selection predicates on all attributes) that contain the values of the query object t_q, and report the subspace where the query object has the highest percentile rank. Algorithm 1 summarizes the details of this method. Note that hereafter, when referring to group G (e.g., line 10 of Algorithm 1), we mean the relation after applying all selected predicates so far. Also we define the *quality* $qual(G)$ of G as the ratio of all tuples in G whose values on attribute $G.M$ are smaller or equal to $t_q.M$. Obviously, $qual(G) = pr(t_q, G)$.

Time complexity. The worst-case complexity of the Naive Algorithm is $O(|R| \cdot \Pi_{i=1}^{m_o} \omega_i^2)$. There are $O(\Pi_{i=1}^{m_o} \omega_i^2)$ subspaces that contain t_q, where m_o is the number of predicate attributes and in the worst case all predicate attributes are ordinal and predicate A_i contains $O(\omega_i^2)$ ranges ($\omega_i = \min\{|R|, |D_i|\}$). For each subspace, we need $O(|R|)$ time to compute the percentile rank of t_q.

Algorithm 2 Materialization Algorithm

```
1: bestrank := qual(R);
2: bestG := R; bestPreds := ∅;
3: for each subspace S_i ∈ S do
4:     ub_i ← (|S_i| − φ_j)/|S_i|, where j satisfies
           F^i_{φ_j} > t_q.M ≥ F^i_{φ_{j+1}};
5:     lb_i ← (|S_i| − φ_j + 2)/|S_i|, where j satisfies
           F^i_{φ_{j−1}} ≥ t_q.M > F^i_{φ_j};
6: δ := the greatest lb_i for 1 ≤ i ≤ |S|;
7: S* ← {S_i | ub_i > max(δ, bestrank)};
8: for each unpruned subspace S_i ∈ S* do
9:     Derive complete S_i from its predicates Preds;
10:    if qual(S_i) > bestrank then
11:        bestrank := qual(S_i);
12:        bestG := S_i; bestPreds := Preds;
13: return {bestG, bestPreds};
```

An alternative to this naive approach would be to adopt the solution framework of [15]. The basic idea is to sample the scores of records at all subspaces and derive upper and lower bounds for the ranking of objects there. More precisely, let $F_1^i, F_2^i, \ldots, F_n^i$ be the complete ranked list of object scores in a subspace S_i, and ϕ_1, ϕ_2, $\ldots$, ϕ_l $(1 \leq l \leq n)$ be a sequence of distinct (sampled) positions, such that $1 \leq \phi_j \leq n$ and $\phi_i < \phi_j$ if $i < j$. The scores $F_{\phi_1}^i, F_{\phi_2}^i, \ldots, F_{\phi_l}^i$ are materialized for subspace S_i. Obviously all the scores are materialized in S_i when $l = n$. Given a query object t_q, if $F_{\phi_j}^i > t_q.M$, we can conclude that the rank of t_q in S_i should be larger than ϕ_j; if $F_{\phi_j}^i < t_q.M$, the rank of t_q should be smaller than ϕ_j. Based on this observation, we can solve SMP, as illustrated by Algorithm 2.

Let $\mathcal{S}$ be the set of all subspaces in the input relation, which include t_q; Algorithm 2 first considers each subspace S_i in $\mathcal{S}$ and calculates the upper and lower bound of the ranking that the query tuple t_q has in S_i. Then, threshold δ is computed as the minimum possible percentile rank of t_q in any subspace (line 6). All subspaces whose upper bounds are no greater than $\max(\delta, bestrank)$ are pruned, because they cannot be the optimal subspace. Finally, each remaining subspace

$S_i \in \mathcal{S}^*$ is verified by computing the exact percentile rank of t_q in it and the one wherein t_q has the best percentile rank is output as the SMP result.

Although the materialization strategy greatly reduces the number of subspaces to be considered, it is still costly to verify the exact percentile rank in the remaining subspaces. Besides, the pruning power of the materialization solution heavily depends on the sampled scores in each subspace; i.e., there does not exist a sampled position sequence that is optimal for all possible query tuples t_q.

Time complexity. Computing the lower/upper bound of t_q at each subspace (lines 4 and 5) takes $O(\log l)$ time using binary search; after that, we apply for each unpruned subspace the corresponding predicates to derive the result group in $O(|m_o| \cdot |R|)$ time and use additional $O(|R|)$ time to compute the percentile rank. Thus, the overall time complexity is $O(|\mathcal{S}| \cdot \log l + |\mathcal{S}| + |\mathcal{S}^*| \cdot (m_o \cdot |R| + |R|))$, i.e., $O(\log l \cdot |\mathcal{S}| + m_o \cdot |R| \cdot |\mathcal{S}^*|)$, where $|\mathcal{S}| = \Pi_{i=1}^{m_o} \omega_i^2$.

3.3 Single-path Browsing Algorithm

In view of the hardness of SMP, we propose a number of greedy approaches which compute an approximate solution. Our first method, called *browsing algorithm* (BA), is inspired by rule-based classifiers [3], which extract classification rules from a set of training records with positive/negative labels. BA adapts and optimizes the greedy rule selection approach, by iteratively selecting on each attribute the domain subrange which (i) includes t_q, (ii) includes at least $sup \cdot |R|$ records when applied together with the predicates selected so far, where sup is the minimum support constraint, and (iii) maximizes the ratio of positive to all tuples covered by the rule (i.e., range). A tuple is labeled *positive* (*negative*) if it ranks lower than or equal to t_q (higher than t_q). BA takes its name from the fact that it browses the space around t_q in all dimensions, in order to find the best multidimensional range that includes t_q.

Algorithm 3 BA algorithm

```
1: G := R; Preds := ∅; bestrank := qual(R); pred := none;
2: for each attribute A not in Preds do
3:     A.pred:=best predicate on A for G
4:         such that t_q.A ∈ A.pred and |σ_{A.pred}G)| ≥ sup · |R|;
5:     if qual(σ_{A.pred}G) > bestrank then
6:         pred := A.pred;
7:         bestrank := qual(σ_{A.pred}G);
8: if pred=none then
9:     return {G, Preds};
10: else
11:     Preds := Preds ∪ {pred};
12:     G := σ_{pred}G;
13:     goto Line 2;
```

Algorithm 3 is a high-level description of BA, which takes a group of tuples G (initialized to the complete set of tuples in R) and at each step computes a subset of G that includes t_q by applying a new predicate.The set of predicates selected so far is recorded in $Preds$. In the main loop (Lines 2–7), for each attribute A for which we do not have a predicate in $Preds$ yet, we find the *best* predicate $A.pred$; i.e., the one that maximizes $qual(\sigma_{A.pred}G)$ among all that qualify $t_q \in \sigma_{A.pred}G$ and $|\sigma_{A.pred}G| \geq sup \cdot |R|$. Lines 5–7 check whether $A.pred$ is better than the best predicate $pred$ found so far and updates $pred$ if so. After all attributes have been examined, if no predicate is found that can improve the current G (Line 8), the algorithm terminates reporting $Preds$ as the final set of predicates. Otherwise (Lines 11–13), the best predicate found $pred$ is added to the current set $Preds$ and G is restricted to the set of tuples after applying $pred$ on G. Then, BA tries to find another predicate to further improve G if applicable.

Time complexity. At each iteration, BA conducts for each remaining attribute A_i a counting sort to compute the best predicate on this attribute in $O(|R| + \omega_i)$ time (as we shall discuss in Section 3.3.1); at the same time BA picks the best attribute. In the worst case, BA considers all the remaining attributes at each iteration. Thus, the overall cost of BA is $O(m_o^2 \cdot (|R| + \omega))$, where $\omega = \max\{\omega_1, \cdots, \omega_{m_o}\}$. Since $\omega_i = \min\{|R|, |D_i|\}$, the overall cost of BA can be simplified as $O(m_o^2 \cdot |R|)$.

Example 3: Consider Table 2 and let sup = 0.3 and t_q = o_6 be the query object. In the first BA round, $G = R = \{o_1, o_2, \ldots, o_{10}\}$ and $bestrank = 50\%$. The best predicates and the corresponding percentile ranks of o_6 on attributes `MPG`, `Year` and `HP` are (27, 40] (66.67%), (2006, 2010] (66.67%) and $(-\infty, 138]$ (57.14%), respectively. Since the percentile rank of o_6 is maximized when using (27,40] as a range predicate on `MPG`, BA selects this range and adds it to the predicates set $Preds$. Now G becomes $\{o_3, o_6, o_8\}$. Since there are only 3 objects in G, i.e., as many as the minimum support threshold $sup \cdot |R| = 0.3 \cdot 10 = 3$, BA in the next loop cannot find any other predicate that further improves the quality of G, therefore it terminates and reports $\{$`MPG`: $(27, 40]\}$.

3.3.1 Predicate selection on a single attribute

A naive way to select the best *range predicate* on a single attribute A (i.e., lines 3–4 of Algorithm 3) is to enumerate all ranges that include $t_q.A$, which bears a cost quadratic to the domain size of A. We now propose a module which performs the same task in time linear to $\min\{|G|, D_A\}$, provided that the records in G have been ordered based on their A-values. Our solution considers an ordered projection $G.A$ of G's tuples on A. Assume that $G.A$ contains D_A distinct values $v_1, v_2, \ldots, v_{D_A}$. Without loss of generality, let $v_i < v_j$ if $i < j$. In addition, we assume that there is a dummy value $v_0 = -\infty$ in the value domain. Let G_{ij} $(0 \leq i \leq j)$ be the set of objects in G whose A values are in range $(v_i, v_j]$. In order to find the best percentile rank of the query object t_q, we build two *counting arrays* C and D on the domain values of $G.A$. C_i is the number of objects in G_{0i} (i.e., $C_i = |G_{0i}|$), and D_i is the number of objects in G_{0i} with rank lower than or equal to t_q (obviously, $C_0 = D_0 = 0$).

Given a range $(v_i, v_j]$ (i.e., $[v_{i+1}, v_j]$) covering t_q, the percentile rank of t_q in G_{ij} can be easily derived as $(D_j - D_i)/(C_j - C_i)$.

Example 4: Assume that G is the relation shown in Table 2 and let sup=0.3 and $t_q = o_6$. Consider the ordered projections of the tuples on MPG, as shown in the second line of Table 3; t_q.MPG = v_6 = 34 is shown in bold font. The C and D values for every distinct value v_i in the domain of the MPG attribute are computed by considering the tuples in G in increasing MPG order.

TABLE 3
Example of finding the best predicate

D_{MPG}	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
$G.MPG$	-∞	18	20	22	25	27	30	**34**	40
C	0	1	2	3	6	7	8	9	10
D	0	1	1	1	3	3	3	4	5

Lemma 1: If (C_i, D_i) and (C_j, D_j) are viewed as two points in the 2D Euclidean space, the percentile rank of t_q in the range $(v_i,v_j]$ equals the slope of the line that passes through (C_i,D_i) and (C_j,D_j).

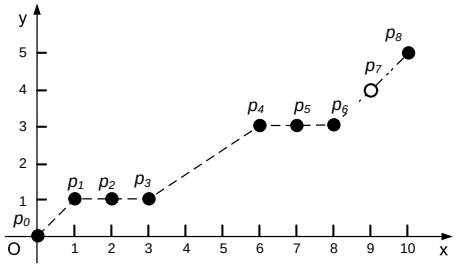


Fig. 1. Points from Example 4 in Euclidean space.

Our solution is built based on Lemma 1. For each distinct value v_i, we define a point $p_i(C_i, D_i)$ in the Euclidean space. Let $v_k = t_q.A$, the original problem now becomes finding two points p_b and p_e ($s < k \leq e$) such that the slope of line p_bp_e is maximized and the length of the horizontal projection of p_bp_e is not less than $sup \cdot |R|$. Figure 1 shows the mapping of the C and D values in a Euclidean space with $x = C$ and $y = D$. From the ranges covering t_q.MPG=34 (corresponding to point p_7) and containing at least $sup \times |R|$ = 3 points, the one with the largest slope is $(v_5, v_8]$, corresponding to points p_5 and p_8. We now show some properties that serve as building blocks to our solution.

Lemma 2: Consider three candidates p_{b_1}, p_{b_2} and p_{b_3} for forming the line with the largest slope with a given right-end point p_e ($b_1 < b_2 < b_3 < e$). If the slope of $p_{b_2}p_{b_3}$ is smaller than that of $p_{b_1}p_{b_2}$, then $p_{b_2}p_e$ cannot be the line with the largest slope.

Proof: Consider three points p_{b_1}, p_{b_2} and p_{b_3} ($b_1 < b_2 < b_3$) as shown in Figure 2; the slope of $p_{b_2}p_{b_3}$ is smaller than the slope of $p_{b_1}p_{b_2}$. Given a fixed right endpoint p_e ($e > b_3$): (i) the slope of line $p_{b_2}p_e$ is no less than that of $p_{b_1}p_e$ only when p_e lies in the area above line $p_{b_1}p_{b_2}$; (ii) the slope of line $p_{b_2}p_e$ is no less than that

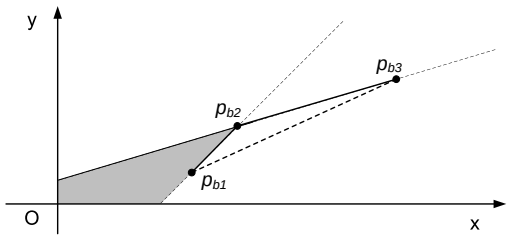


Fig. 2. Pruning case for left endpoint

of $p_{b_3}p_e$ only when p_e lies in the area below line $p_{b_2}p_{b_3}$. Therefore, the slope of $p_{b_2}p_e$ is no less than the slopes of both $p_{b_1}p_e$ and $p_{b_3}p_e$ if and only if p_e lies in the overlap area of the aforementioned two ones (the shaded area), i.e., $e < b_2$. However, this contradicts our assumption that $e > b_3$; therefore $p_{b_2}p_e$ cannot be the line ending at p_e with the largest slope. □

For a given right endpoint p_e, for which we search for the left endpoint p_b, such that the slope of p_bp_e is maximized, we can use Lemma 2 to prune all points p_{b_2} before p_e for which there is a preceding point p_{b_1} and a succeeding point p_{b_3} satisfying the condition of the lemma. The remaining left endpoint candidates (in sequence) form a polyline L_e; the slopes of consecutive line segments in L_e are monotonically increasing (i.e., L_e is convex). More importantly, as Lemma 2 suggests, pruning left endpoint candidates to form polyline L_e is actually independent of p_e; two different right endpoints can share common left endpoint candidates. Particularly, given two right endpoint candidates p_e and $p_{e'}$ ($e < e'$), the left endpoint candidates sequence in L_e is a prefix of that in $L_{e'}$. The following lemma can be used to avoid redundant computations, while searching for the left endpoint for a right endpoint $p_{e'}$, based on the best left endpoint for a previous right endpoint p_e ($e < e'$).

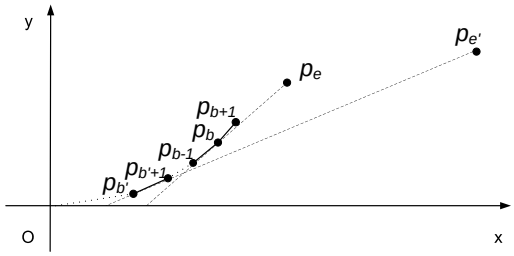


Fig. 3. Best left endpoint

Lemma 3: Consider two right endpoints p_e and $p_{e'}$ ($e < e'$). Assume that the best left endpoints for p_e and $p_{e'}$ are p_b and $p_{b'}$, respectively. If $p_{b'}p_{e'}$ has a greater slope than p_bp_e, then $b' \geq b$.

Proof: (By contradiction). Consider p_e and $p_{e'}$ as shown in Figure 3 and assume that there exists some left endpoint $p_{b'}$ ($b' < b$), which is the best left endpoint for $p_{e'}$ and the slope of $p_{b'}p_{e'}$ is greater than that of p_bp_e. Since p_b is the best left endpoint for p_e, the slope of p_bp_e

should not be smaller than that of line $p_{b-1}p_b$, otherwise p_{b-1} will contribute a larger slope w.r.t. p_e. On the other hand, since $p_{b'}$ is the best left endpoint for $p_{e'}$, the slope of $p_{b'}p_{e'}$ should not be greater than that of line $p_{b'}p_{b'+1}$. However, due to the fact that the slopes of consecutive line segments in L_p are non-decreasing, we know that the slope of $p_{b'}p_{b'+1}$ is less than or equal to that of $p_{b-1}p_b$. Therefore, the slope of $p_{b'}p_{e'}$ is smaller than the slope of p_bp_e, which is a contradiction. □

Algorithm 4 Finding the best predicate on attribute A

```
1: bestrank := −∞;
2: l := 0;                    ▷ v_l is first possible left-end value (dummy)
3: r := index of first possible right-end value;    ▷ v_r = t_q.A
4: Q := {};                   ▷ double-ended queue of left endpoints
5: while r ≤ D_A do           ▷ v_r is a valid right-end value
6:     if (v_l, v_r] satisfies the support constraint then
7:         while |Q| ≥ 2 and prune(Q_{first−1}, Q_first, p_l) do
8:             dequeue Q_first;
9:         enqueue p_l at the front of Q;
10:        l := l + 1;
11:    else
12:        while |Q| ≥ 2 and slope(Q_{last−1}, p_r) > slope(Q_last, p_r) do
13:            dequeue Q_last;
14:        if |Q| > 0 and slope(Q_last, p_r) > bestrank then
15:            bestrank := slope(Q_last, p_r);
16:            bestrange := value range of (Q_last, p_r);
17:        r := r + 1;
18: return {bestrange, bestrank};
```

Based on Lemmas 2 and 3, Algorithm 4 enumerates all possible right endpoints in the outer loop (line 5) in sequence (the first possible right endpoint has the value $t_q.A$). For each right endpoint p_r, it maintains a polyline L_r as a double-ended queue Q, which contains the candidate left points p_l for the line segment p_lp_r with the largest slope (the last element of Q is the leftmost point in L_r). While considering left endpoints to be added to Q, the algorithm checks whether the next left endpoint p_l can be used to prune the first element of Q using Lemma 2 (line 7). As soon as no more left endpoints can be considered for p_r (due to the support constraint), the contents of Q (i.e., the polyline L_r) are examined in order (from left to right) to find the left endpoint that forms the largest slope with p_r; the contents of Q which have a smaller slope with p_r are removed from its tail, since, according to Lemma 3, they cannot contribute larger slopes with the next right endpoints to be considered (line 13). Then, the algorithm checks whether largest slope for p_r corresponds to the best range for A found so far and updates the best range in this case (lines 14–16). In the rest of the paper, when we say a predicate "satisfies the support constraint" (e.g., line 6), we mean that the predicate satisfies the constraint *together with all other predicates selected so far*.

Example 5: Consider the problem described in Example 4 and the mapping of the C and D arrays to points, as shown in Figure 1. In the beginning, $r = 7$ ($v_r = 34$), and p_0, p_1 are enqueued into Q successively since both $(v_0, v_7]$ and $(v_1, v_7]$ satisfy the support threshold. The next point p_2 also satisfies the support constraint (range $(v_2, v_7]$ contains enough tuples); however, it forms a pruning case with the previous two objects in Q, i.e., p_0 and p_1. As a result, p_1 is dequeued from Q (line 8) and p_2 is enqueued. Now $Q = \{p_0, p_2\}$. Similarly, p_2 is pruned by p_3 in the next step, resulting in $Q = \{p_0, p_3\}$. The next point p_4 corresponds to a valid range $(v_4, v_7]$; in addition, p_0, p_3 and p_4 do not form a pruning case. Thus, p_4 is simply enqueued into Q, i.e., $Q = \{p_0, p_3, p_4\}$. Since $(v_5, v_7]$ contains fewer objects than the support threshold, we stop the enumeration of left endpoints for p_7, and move on to the calculation of the best slope for segments ending at p_7 (lines 11–17). The line with the best slope (50%) is p_3p_7 and the corresponding range $(v_3, v_7]$ is recorded as the current best for A. Note that p_0 is dequeued from Q according to Lemma 3. Then, we move on to the next right endpoint (line 17), which is p_8, and continue examining left endpoints from where we stopped before ($p_l = p_5$). Range $(v_5, v_8]$ contains enough tuples. However, p_3, p_4 and p_5 form a pruning case; therefore, p_4 is dequeued from Q and p_5 is enqueued into Q, i.e., $Q = \{p_3, p_5\}$. Next, we consider p_6; since $(v_6, v_8]$ does not satisfy the support constraint, we stop and find the best range ending at v_8, which is $(v_5, v_8]$ with slope 66.67%. Since there are no more right-endpoint candidates, the algorithm reports $(v_5, v_8]$ as the best predicate for attribute MPG.

Lemma 4: Algorithm 4 finds the range covering t_q which determines the best percentile rank for t_q at time complexity $O(w_i)$, assuming that the values of $G.A_i$ have been ordered.

Proof: Observe that the operations on Q are only enqueuing and dequeuing operations that take $O(1)$ time. Each left endpoint may only be added to Q once (after been dequeued from Q it is never enqueued again). Since we have $O(w_i)$ endpoints in total, the time spent on queuing operations is $O(w_i)$. For each iteration in the outer loop, either l or r are increased by one, which means that after at most $O(w_i)$ iterations, the outer loop will terminate. Thus, the total time spent for the operations at line 10 and lines 14–17 is $O(w_i)$. □

3.4 Diversified-path Browsing Algorithm

Although BA is very fast, it may not find a solution with quality close to the optimal, because it considers and fixes for each attribute only one range (i.e., the one of maximum quality). This constrains the possible choices for the following attributes and the search space considered by BA. In this section, we propose a more relaxed variant of BA, which explores a larger portion of the solution space. This *Diversified-path Browsing Algorithm* (DBA) selects multiple *diversified* range predicates for each attribute, in order to explore diverse parts of the solution space, while being fast. Before going into the details of DBA, we first discuss how to define and compute a set of diversified predicates for a single

Algorithm 5 Finding DP and best percentile rank

```
   H := ∅                          ▷ keeps all DP indexed by left endpoint
1: Lines 1 to 16 of Algorithm 4
2: ...
3:     if H_{Q_last} already exists
4:         if slope(H_{Q_last}) < slope(Q_last, p_r) then
5:             H_{Q_last} := (Q_last, p_r];
6:     else
7:         insert (Q_last, p_r] to H;
8:     r := r + 1;
9: return H and {bestrange, bestrank};
```

attribute. We define a dominance relationship between range predicates on the same attribute, as follows.

Definition 1 (p-Dominance): Consider two range predicates $[b_1, e_1]$ and $[b_2, e_2]$ on attribute A; $[b_1, e_1]$ is p-dominated (predicate-dominated) by $[b_2, e_2]$ iff either $b_1 = b_2$ or $e_1 = e_2$, and the quality of $[b_2, e_2]$ is larger than the quality of $[b_1, e_1]$ for t_q.

Recall that the quality of an attribute range is defined by the percentile rank of t_q in it. Based on Definition 1, if, on some attribute A, range predicate α is p-dominated by another predicate β, then β is considered to be a more *representative* predicate for A, because the query object is more highly ranked in β, and α, β cover similar sets of objects. Therefore, predicates that are not p-dominated by any other predicate are intuitively the most diversified and representative predicates on A. The set of *Diversified Predicates* (DP) for a given attribute A are formally defined as follows.

Definition 2 (Diversified Predicates (DP)): Consider the set of valid ranges W on a single attribute A (i.e., those that include t_q and satisfy the support threshold together with all selected predicates so far). The Diversified Predicates (DP) of A are all ranges in W that are not p-dominated by any other range in W.

Now we show how we can modify Algorithm 4 to find DP for an attribute A, without adding extra time complexity. Algorithm 5 extends Algorithm 4 as follows. For each right endpoint p_r, after finding the best range $(p_l, p_r]$ considering the current contents of Q as candidate left endpoints, Algorithm 5 adds $(p_l, p_r]$ to a hash table H, which uses the left endpoint of the ranges as a key. If there is another range in H with the same left endpoint already in H, then it is compared with the new range and the best of the two is kept in H only (lines 3–5). This guarantees that no ranges in H share identical left endpoints (and that H keeps only the range of best quality among those that share the same left endpoint). In addition, since only one range is selected for each right endpoint, there are no two ranges in H that have the same right endpoints. The following lemma proves that the hash table H after a run of Algorithm 5 has essentially the same content as DP.

Lemma 5: Algorithm 5 finds exactly the correct DP.

Proof: We first denote the set of all *valid* ranges for attribute A (i.e., those that include $t_q.A$ and qualify the support threshold) as W. For each right endpoint p_{r_i}, let $p_{l_i}p_{r_i}$ be the segment which has the largest slope among all $p_l p_{r_i} \in W$. Let W_q be the set of all $p_{l_i}p_{r_i}$ segments. Obviously, for any range x, such that $x \in W$ and $x \notin W_q$, x could not be in the DP of A, since x must be p-dominated by some range in W_q. Thus the DP of A must be a subset of W_q. Assume all the ranges found by Algorithm 5 is W_o. We now prove that any segment (i.e., range) x in $W_q - W_o$ must be p-dominated by some other segment, and thus not included in DP. Consider such a range x and let p_{l_x} and p_{r_x} be its left and right endpoints, respectively. (i) If there is another range $x' \in W_o$, such that $p_{l_x} = p_{l_{x'}}$ (i.e., x and x' share the same left endpoint), then x' must p-dominate x because no two ranges with the same left endpoint can be in H. (ii) If there is no other range $x' \in W_o$, such that $p_{l_x} = p_{l_{x'}}$, then this means that the best slope for p_{r_x} found by Algorithm 5 (i.e., also by Algorithm 4) is not $p_{l_x}p_{r_x}$, because p_{l_x} has been pruned due to Lemma 3. In other words, the best slope for p_{r_x} found is $p_{l'_x}p_{r_x}$, such that $l'_x > l_x$. Since $p_{l'_x}$ is surely found as a best left point for some right point before encountering p_{r_x} (consider line 4). Without loss of generality, let us assume such a right endpoint is $p_{r'_x}$ ($p_{r'_x} < p_{r_x}$). Based on Lemma 3, given $l'_x > l_x$, we can prove by contradiction that the slope of $p_{l_x}p_{r_x}$ is not greater than $p_{l_x}p_{r'_x}$, i.e., $p_{l_x}p_{r_x}$ is p-dominated by $p_{l_x}p_{r'_x}$. Therefore, in both cases x is p-dominated. Similar to (ii), we can prove that any range in $W_o - W_q$ must be p-dominated by some other range. Thereby, considering also the fact that any two ranges in the output H of Algorithm 5 do not share the same left or right endpoints, H has exactly the same content as the DP of A. □

Attribute Pruning. When choosing a range predicate on an attribute, a subset of records is selected to become the group G of records to be considered for the next attribute. Let $pos(G)$ be the number of tuples in G that are not better than t_q w.r.t. the measure attribute M. Note that if $pos(G) < sup \cdot |R|$, then the maximum possible percentile rank of t_q in G regardless the additional selection predicates to be applied on G is $qual^+(G) = pos(G)/(sup \cdot |R|)$, i.e., less than 100%. While browsing the search space, DBA uses this *upper quality bound* $qual^+(G)$ to avoid applying additional predicates on a set G, if $qual^+(G)$ is less than the quality of the best predicate set found so far.

Algorithm 6 is a pseudocode for DBA, which extends BA to consider multiple subspaces based on diversified predicates. The algorithm in lines 4–5, like BA, iteratively picks the attribute A with the overall best predicate $A.pred$ among all attributes on the input set G (initially $G = R$). If it can no longer find any other predicate which would satisfy the minimum support threshold, it checks whether the current set of selected attributes $Preds$ is the best set found so far (lines 6–10). Otherwise, the set of diversified predicates for A is computed using Algorithm 5 and for each such predicate $pred'$, if not pruned using the upper quality bound $qual^+$ (line 15),

we (i) append $pred'$ to $Preds$ and create a extended predicates set $Preds'$ and (ii) recursively run DBA after applying $Preds'$ on G (line 19). Thus DBA operates like running multiple instances of BA, one for each combination of diversified predicates for all attributes taken in a specific order.

Algorithm 6 DBA algorithm

```
1: G_best := r; Preds_best := ∅; bestrank := qual(G);
2: procedure DIVERSERANGE(G, Preds)
3:     pred := none;
4:     Lines 2 to 6 of Algorithm 3
5:     ...
6:     if pred = none then
7:         if qual(σ_Preds G) > bestrank then
8:             bestrank := qual(σ_Preds G);
9:             G_best := G;
10:            Preds_best := Preds;
11:    else
12:        A := the attribute of pred;
13:        DP_A := Diversified Predicates on A of G;
14:        for each predicate pred' ∈ DP_A do
15:            if qual+(σ_pred' G) ≤ bestrank then
16:                continue;
17:            Preds' := Preds ∪ {pred'};
18:            G' := σ_pred' G;
19:            DIVERSERANGE(G', Preds');
20: return {G_best, Preds_best}
```

Note that the number of diversified predicates for an attribute could be too few or too many, a fact that makes the performance and cost of DBA unstable. In order to control the cost and the exploration power of the algorithm, we introduce a parameter k, which determines the number of diversified predicates to use per attribute. If DP for a given attribute is larger than k, then we simply use the k predicates in DP with the largest slopes. If DP is smaller than k, then we add back to DP p-dominated predicates which are pruned from the hash table H during Algorithm 5 (line 4). For this purpose, Algorithm 5 is adjusted to maintain in a heap the best k pruned predicates and in the end add back to DP the $k-|\text{DP}|$ pruned predicates of the highest quality.

Time complexity. DBA differs from BA in that it chooses k diversified predicates at each iteration for each attribute. There are at most m_o iterations, thus the overall cost of DBA is $O(m_o^2 \cdot |R| \cdot k^{m_o})$.

Example 6: Consider again Example 3 and that we run DBA for query object o_6 and $k = 2$. In the first round, the percentile rank of o_6 is maximized when using $(27, 40]$ as a range predicate on `MPG`. Therefore, we select `MPG` and find the diversified predicates on `MPG`, which are ranges $(27, 40]$ (66.67%) and $(22, 34]$ (50%). Then, for each predicate, we create an instance G' of G by applying the predicate and continue the diversified predicates selection recursively. Figure 4 illustrates the running procedure of DBA by showing the diversified predicates chosen at each round. The objects in the node are sorted according to their corresponding attribute values. The range predicate chosen at each node is underlined, and

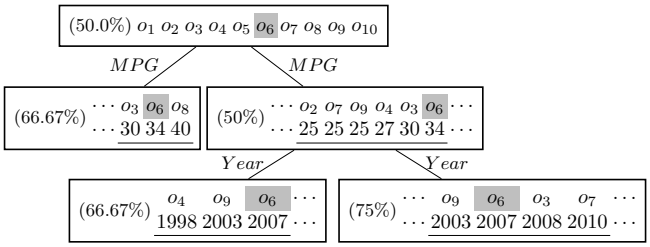


Fig. 4. Diversified predicates selected by DBA

the corresponding percentile rank is given for reference as well. At each leaf of the tree, DBA cannot find an additional predicate from the remaining attributes and enters lines 6–10. The best overall predicate-set found corresponds to the best path, i.e., {`MPG`: (22,34], `Year`: (2000, 2010]} with percentile rank 75%.

3.5 Enumerating Diversified-path Browsing

BA and DBA efficiently identify an approximate solution to SMP. However, like rule-based classifiers, the attributes are considered and examined in a *single* order, determined by our greedy strategy. For example, BA iteratively picks the most promising attribute at each iteration by applying Algorithm 4 on all remaining attributes. In this section, we introduce an approach which has increased probability of finding a better percentile rank. We extend DBA to an *Enumerating Diversified Browsing Algorithm* (EDBA), which examines all possible permutations of predicate attributes in order to find a better approximate percentile rank for the query tuple.

Algorithm 7 EDBA algorithm

```
1: G := r; Preds = ∅; bestrank := qual(G);
2: bestG := G; bestPreds := none;
3: procedure ENUMATTRIBUTE(G, Preds)
4:     if all attributes are in Preds then
5:         if qual(G) > bestrank then
6:             bestrank := qual(G);
7:             bestG := G; bestPreds := Preds;
8:     else
9:         Q := {}                                  ▷ priority queue
10:        for each attribute A not in Preds do
11:            A.pred:=best predicate on A for G such that
12:                t_q.A ∈ A.pred and |σ_A.pred G)| ≥ sup · |R|;
13:            Insert A into Q with priority qual(σ_A.pred G);
14:        while Q is not empty do
15:            A := next attribute in Q;
16:            Lines 13 to 18 of Algorithm 6
17:            ...
18:                ENUMATTRIBUTE(G', Preds');
19: return {bestG, bestPreds};
```

Instead of selecting the next attribute to choose at each level of the search tree in random order, EDBA (Algorithm 7) prioritizes the available attributes according to their improvement $qual(\sigma_{A.pred}G)$ to the current percentile rank $qual(G)$. More specifically, we prioritize the attributes at each level, based on how much improvement they give to the current percentile of t_q, according

to Algorithm 4. This way, we maximize the probability of finding a good approximate solution $Preds_{best}$ to SMP early, which can be used to prune more predicates and search subtrees, based on the $qual^+$ bound.

Time complexity. EDBA extends DBA by examining all possible permutations of predicate attributes, instead of picking the remaining attribute greedily one by one. Therefore, EDBA takes $O(m_o! \cdot |R| \cdot k^{m_o})$ time.

4 Experimental Evaluation

In this section, we experimentally evaluate the efficiency and effectiveness of our proposed solutions to SMP. All methods were implemented in C++ and the experiments were conducted on a 3.40 GHz quad-core machine running Ubuntu 12.04, with 16 GB of main memory. We use three real datasets in our evaluation:

NBA: We crawled information about 24,524 NBA players and their performance per season up to 2012-2013 from NBA & ABA Basketball Statistics[1]. For each tuple, `Average Points` is used as the measuring attribute; 3 ordinal attributes `Height`, `Weight` and `Born`, which are not correlated to the measuring attribute are extracted and used as selection attributes. The domain sizes (i.e., the number of distinct values) of these attributes are 28, 142 and 83 respectively.

SONG: We crawled information about 306,297 songs from the "Million Song Dataset"[2]. The `hotness` of a song is used as the measuring attribute; 5 additional ordinal attributes: `Year`, `Tempo`, `Segments Pitches`, `Loudness` and `Duration` are extracted as selection attributes. The domain sizes of these attributes are 85, 261, 216, 57 and 1357 respectively.

PAM: This dataset contains records of different human physical activities (e.g., walking, cycling, etc.) from the PAMAP project[3]. There are 1M tuples in this dataset. We chose 8 ordinal selection attributes corresponding to statistical measures collected from sensors (e.g., `acceleration`, `magnetometer`, `gyroscope`, etc.) and 1 measuring attribute (i.e., `temperature`). The domain sizes of the 8 selection attributes are 796, 844, 835, 860, 851, 848, 830, and 834, respectively.

In our comparison, we include our three greedy methods (BA, DBA, and EDBA) and the NAIVE algorithm presented in Section 3.2, which finds the exact solution to SMP but has high cost. We do not include the Materialization algorithm (MA) [15] (also reviewed in Section 3.2) because we found that it is 2-3 orders of magnitude more expensive than NAIVE and it cannot terminate within reasonable time for our datasets, where the attributes have relatively large domain sizes. As an indication, Table 4 compares NAIVE with MA on our smallest dataset (NBA), using only two attributes (`Weight` and `Born`) for various values of sup.

1. http://www.basketball-reference.com
2. http://labrosa.ee.columbia.edu/millionsong
3. http://www.pamap.org

TABLE 4
Costs of NAIVE and MA on the NBA dataset

sup	1%	2%	4%	8%	16%
NAIVE	0.465 s	0.467 s	0.467 s	0.469 s	0.472 s
MA	191.15 s	253.87 s	275.13 s	306.82 s	329.96 s

Besides measuring the runtime of all tested methods, we assess our methods based on the quality of the result they find. The query result quality for an algorithm $\mathcal{A}$ is defined as the relative quality of its result compared to the optimal result found by NAIVE. More specifically, given an input object t_q, if the percentile ranks of the subspaces found by algorithm $\mathcal{A}$ and NAIVE are $rank_{\mathcal{A}}$ and $rank_{\mathcal{N}}$, respectively, while the original percentile rank of t_q in the whole dataset is $rank_{\mathcal{O}}$, the quality of $\mathcal{A}$ (on t_q) is defined by:

$$quality(\mathcal{A}) = \frac{rank_{\mathcal{A}} - rank_{\mathcal{O}}}{rank_{\mathcal{N}} - rank_{\mathcal{O}}} \times 100\% \quad (1)$$

TABLE 5
Query parameters

Parameters		Values
k	*NBA*	2, 3, **4**, 5, 6
	SONG	3, 4, **5**, 6, 7
	PAM	4, 5, **6**, 7, 8
sup	*NBA*	1%, 2%, **4%**, 8%, 16%
	SONG	2‰, 4‰, **8‰**, 16‰, 32‰
	PAM	1‰, 2‰, **4‰**, 8‰, 16‰

For each experimental instance (e.g., for a given dataset and values of k and sup), we run 100 SMP queries choosing t_q randomly from the corresponding dataset and average the query execution time and query result quality. Table 5 summarizes the query parameters used in our experiments, with the default values in bold. Recall that k is a parameter to control the number of p-dominated predicates to be used per attribute in DBA and EDBA, while sup is the support threshold in SMP. Note that the sup values used for the NBA dataset are greater than those used for SONG and PAM, because the last two datasets are much larger and even a small support threshold will form a significant peer group.

4.1 Case Study

We first conduct a case study on NBA that illustrates the usefulness of SMP. Table 6 shows a sample of query objects, their original ranks in the dataset, and the predicates that maximize their standings (i.e., found by NAIVE) together with the achieved ranking by them. We also show the predicates found by running our three methods BA, DBA, and EDBA, with corresponding promoted ranks by them. The sup is set to 4% in all cases.

For example, we used tuple t_q = "Yao Ming (2006)" as a query, which is globally ranked 375th out of 24524 tuples (i.e., percentile rank 98.5%). Using NAIVE and our proposed solutions DBA and EDBA, we can find a

TABLE 6
A case study on the NBA dataset

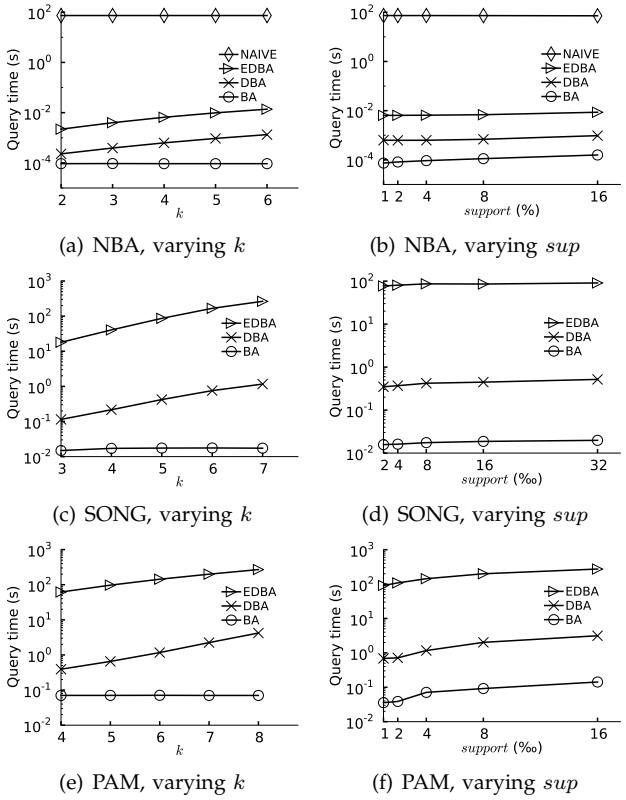
Query object	Object description & original rank	Predicate-sets and promoted ranks			
		NAIVE	BA	DBA	EDBA
Yao Ming (2006)	Height = 7′6″ Weight = 310 Born = 1980 **375/24524**	Weight: [250, 315] Born: [1964, 1987] **1/1345**	Born: [1979, 1981] **8/1453**	Weight: [250, 315] Born: [1964, 1987] **1/1345**	Weight: [250, 315] Born: [1964, 1987] **1/1345**
Nate "Tiny" Archibald (1971)	Height = 6′1″ Weight = 150 Born = 1948 **17/24524**	Height: $(-\infty, 6'4'')$ **1/7683**	Height: $(-\infty, 6'4'']$ **1/7683**	Height: $(-\infty, 6'4'')$ **1/7683**	Height: $(-\infty, 6'4'')$ **1/7683**
Chris Bosh (2009)	Height = 6′10″ Weight = 228 Born = 1984 **477/24524**	Height: [6′9″, 6′11″] Weight: [225, 245] **1/1822**	Height: [6′10″, 6′11″] Weight: [225, 245] **1/1203**	Height: [6′9″, 6′11″] Weight: [225, 235] **1/1326**	Height: [6′9″, 6′11″] Weight: [225, 245] **1/1822**
Kevin Durant (2012)	Height = 6′9″ Weight = 215 Born = 1988 **141/24524**	Height: [6′9″, 6′12″] Born: $[1964, +\infty)$ **2/4212**	Born: $[1985, +\infty)$ **2/1559**	Born: $[1985, +\infty)$ **2/1559**	Height: [6′9″, 6′12″] Born: $[1964, +\infty)$ **2/4212**
William Sharman (1952)	Height = 6′1″ Weight = 175 Born = 1926 **3001/24524**	Height: [5′11″, $+\infty$) Weight: $(-\infty, 240]$ Born: $(-\infty, 1927]$ **23/1317**	Weight: $(-\infty, 236]$ Born: [1918, 1927] **25/1304**	Weight: $(-\infty, 236]$ Born: [1918, 1927] **25/1304**	Height: [5′11″, $+\infty$) Weight: $(-\infty, 236]$ Born: [1918, 1927] **23/1294**

set of two predicates {Weight: [250, 315], Born: [1964, 1987]}, which lift the query tuple to rank 1st among 1345 qualifying tuples (i.e., a percentile rank 100%). BA fails to find this set of predicates and instead finds {Born: [1979, 1981]}, which lift t_q to rank 8th out of 1453 qualifying tuples (i.e., a percentile rank 99.5%). For t_q ="Nate 'Tiny' Archibald (1971)", all our three methods BA, DBA, and EDBA find the same set of predicates as NAIVE (i.e., the optimal set). The only query where EDBA found a sub-optimal predicate set is "William Sharman (1952)"; still, the quality difference to the set found by NAIVE is tiny.

4.2 Performance Evaluation

We now present experiments on the runtime and quality of all methods. We first show the average running time of SMP queries as a function of k and sup on the three datasets in Figure 5. Note that we exclude the results for NAIVE on the SONG and PAM datasets, because it is several orders of magnitude slower than any of our proposed approaches, and does not terminate within reasonable time. Also note that, on the PAM dataset, we use a cheaper variant of EDBA, which uses k diversified predicates (DP) only on the first two attributes in the enumeration order and just selects the best predicate for the remaining ones. The reason is that PAM has a large number of attributes, so enumerating all orders and all diversified attributes for them renders EDBA too slow to be practical. Besides, according to our tests with the default parameter settings on PAM, using k DP in EDBA on the first two attributes only already gives a satisfactory query quality (83.80% on average compared to the original EDBA, while taking less than 1% of the original EDBA's time). Thus, we use this cheaper version of EDBA in all the experimental instances on PAM.

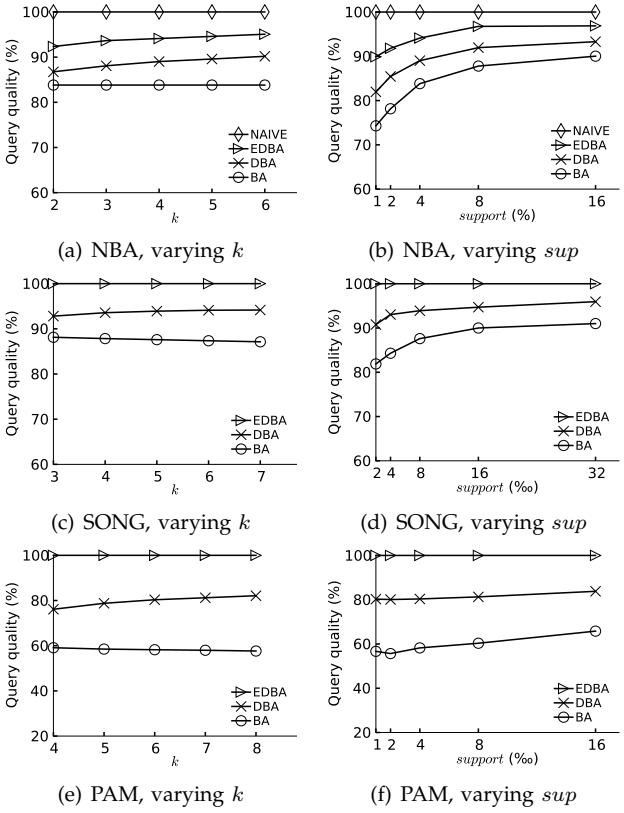
From Figure 5, we observe that our proposed methods are much faster compared to NAIVE on all settings.



(a) NBA, varying k (b) NBA, varying sup
(c) SONG, varying k (d) SONG, varying sup
(e) PAM, varying k (f) PAM, varying sup

Fig. 5. Query time

In addition, as expected, BA is significantly cheaper than DBA, and DBA is significantly cheaper than EDBA. BA's cost is not sensitive to k, because BA just greedily selects one permutation of attributes and one predicate for each of them. DBA multiplies a k^m factor to BA's cost, where m is the number of attributes, because it tries k predicates per attribute. Finally, EDBA has $m!$ times higher cost than DBA because it considers all



(a) NBA, varying k (b) NBA, varying sup

(c) SONG, varying k (d) SONG, varying sup

(e) PAM, varying k (f) PAM, varying sup

Fig. 6. Query quality

permutations of attributes.[4]

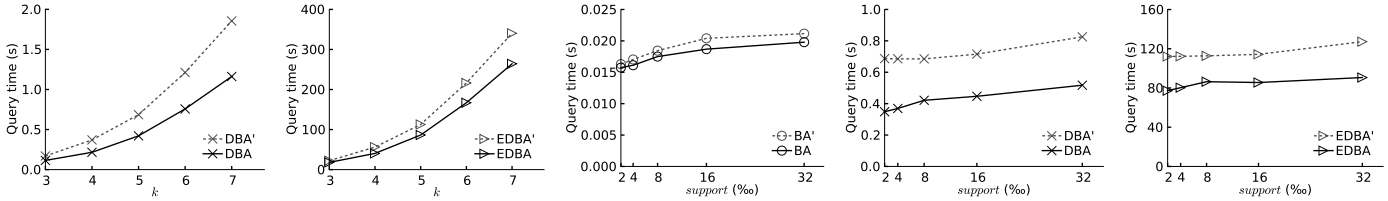
The costs of all methods increase slightly with sup for the following reason. Recall that all our methods use either Algorithm 4 or Algorithm 5 as a module to find the best predicate (or the DP set) for a given attribute; this module is a significant cost factor, because it is called at each node of the search tree. The cost of this module directly depends on the size of its input; first the module has to scan the number of records in the input to construct arrays C and D, and then performs another scan on these arrays to compute its result. Smaller sup values are likely to select predicates at the higher levels of the search tree that are qualified by fewer records compared to large values of sup (recall that $|G|$ should be lower-bounded by $sup \cdot |R|$). Thus, the input size of the predicate search module becomes smaller at the lower levels of the search tree.

Figure 6 plots the average query quality of all methods when varying k and sup, on all three datasets. Note that the exact algorithm NAIVE obtains the optimal solution, so its quality is always 100%. On the other hand, we could only run this algorithm on the NBA dataset, because of its extreme runtime cost on the other two sets. For this reason, in Figures 6(c)–6(f), we consider as optimal the best solution found by any of our methods (i.e., the output of EDBA, which finds the best solution in all cases). BA does not use k, so its quality is not affected by this parameter. As expected, the quality of DBA and EDBA increases with k, because they consider additional diversified predicates on each attribute and therefore explore more search space. Regarding the support threshold sup, when it increases, the search space is decreasing. Thus, all of our methods explore a larger portion of the search space and are more likely to find a solution with quality close to the optimal one.

In Section 3.3.1, we proposed linear-time algorithms (i.e., Algorithms 4 and 5) for finding the best predicate (or diversified predicates) for a given attribute. Now, we study the performance benefits obtained by this module, in practice, when used in our proposed solutions, i.e., BA, DBA, and EDBA. Figures 7 and 8 show the runtime improvement obtained by the linear module compared to using a quadratic algorithm for predicate selection, on SONG and PAM, respectively (NBA is too small and has attributes of small domains). The quadratic algorithm performs a scan on the input to construct arrays C and D (just like Algorithm 4), but then considers all possible ranges on the attribute's domain and uses C and D to measure their quality. BA′, DBA′, and EDBA′ denote the methods where the quadratic predicate selection module was used. Figures 7(a), 7(b), 8(a), 8(b) illustrate the difference between using the linear module and the quadratic module for DBA and EDBA, when varying k (we omit BA because it does not use k). The relative cost difference between DBA and DBA′ (also between EDBA and EDBA′) is maintained for different values of k, as expected, because DBA and DBA′ call the same module the same number of times for a given k.

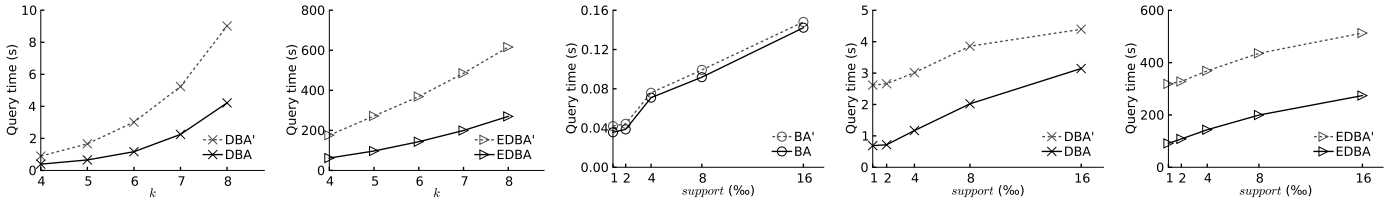
Figures 7(c)–7(e), 8(c)–8(e) show the difference of using the two modules on BA, DBA, and EDBA, on different sup values. First, note that the costs of BA and BA′ are similar. The reason is that BA only explores a single path of the search tree and its cost is dominated by the selection of the first few predicates. For the first predicate, the input size to the predicate selection module is the whole dataset which is very large compared to the attribute domain sizes. Therefore, the construction of arrays C and D dominates the cost of both Algorithm 4 and the quadratic module. DBA and EDBA have a large search tree; therefore the great majority of search tree nodes have a small input size (at the scale of $sup \cdot |R|$), meaning that the cost of preprocessing the input into arrays C and D becomes lower than the cost of computing the DP set at each node, especially by the quadratic algorithm. Therefore, there is a performance gap between Algorithm 5 and the quadratic module in this case. Finally, the cost of all methods is more insensitive to sup on the PAM dataset because PAM contains more attributes (resulting in a larger search tree with more calls to the linear/quadratic module), and the cardinalities of these attributes are large as well (making the performance gap between the linear and quadratic modules more significant).

4. EDBA on PAM uses k DP only for the first two attributes, so its cost difference to DBA is similar with that in the case of SONG.



(a) DBA (k) (b) EDBA (k) (c) BA (sup) (d) DBA (sup) (e) EDBA (sup)

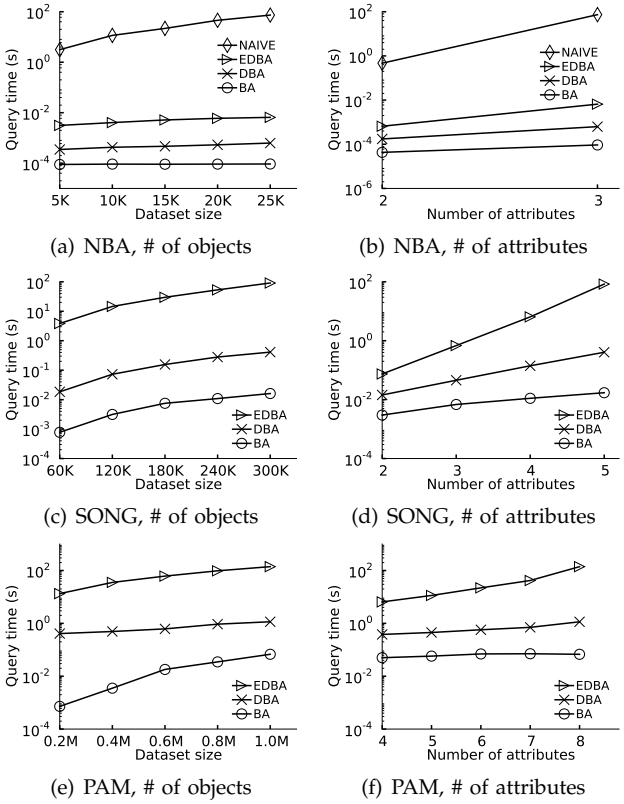
Fig. 7. Comparison of applying different predicate selection modules on SONG



(a) DBA (k) (b) EDBA (k) (c) BA (sup) (d) DBA (sup) (e) EDBA (sup)

Fig. 8. Comparison of applying different predicate selection modules on PAM

4.3 Scalability and Progressiveness



(a) NBA, # of objects (b) NBA, # of attributes

(c) SONG, # of objects (d) SONG, # of attributes

(e) PAM, # of objects (f) PAM, # of attributes

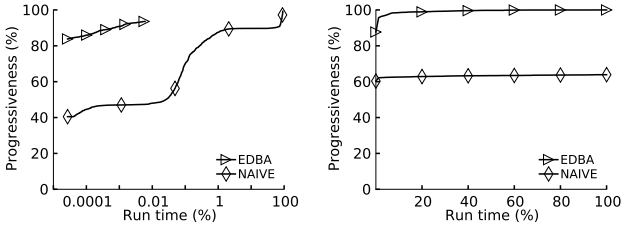
Fig. 9. Scalability tests

Next, we conduct scalability experiments by applying our methods on randomly selected samples of the original datasets of various sizes, which results in a series of datasets having different cardinalities (i.e., 5K–25K records from NBA, 60K–300K from SONG, and 0.2M–1M from PAM). Figures 9(a), 9(c), and 9(e) show the average runtime of tested methods, on subsets of NBA, SONG, and PAM, respectively. Observe that all our methods scale well with the database size, indicating their applicability to larger datasets.

We also study the scalability of our methods to the number of predicate attributes. Again, for each dataset and for each tested value of m (number of attributes), we choose a subset of attributes and apply SMP considering only them. For each m, we choose the m-sized subset attributes with the largest domain sizes from the corresponding dataset. Figures 9(b), 9(d) and 9(f) illustrate the average query time as a function of the number of attributes m on NBA, SONG, and PAM, respectively. BA and DBA are less sensitive to m because they only consider one permutation (i.e., attributes ordering) when selecting predicates. On the other hand, EDBA considers all $m!$ permutations, therefore it does not scale that well. As mentioned in the beginning of Section 4.2, for large values of m, we can use a version of EDBA which selects k diversified predicates only for the first few attributes in each attributes ordering and finds a solution of good quality in practice.

In our last experiment, we assess how fast EDBA converges to a good solution, compared to NAIVE, which does not explore the search space in any particular order and does not consider the best predicates per attribute first. For each method, we plot the average quality of the best solution found as a function of time for 100 queries using the default settings. For queries on the NBA dataset, we let NAIVE to terminate, while for SONG, we stop it at the point where EDBA terminates and plot the quality of its best solution as a ratio of the quality of the best solution found by EDBA.

Figure 10(a) shows that EDBA on NBA converges much faster to a good solution compared to NAIVE and terminates after spending only a small fraction of



(a) NBA (b) SONG

Fig. 10. Progressiveness

NAIVE's time (note that the x-axis is in log-scale). In order for NAIVE to find a solution as good as that of EDBA, it should run about 10^4 times longer. As Figure 10(b) shows, EDBA converges relatively fast to good solution, while NAIVE progresses slowly and the quality of its solutions found early is low. Thus, EDBA can also be used as a *progressive* algorithm that can provide to the user a good solution early and gradually improve it; the user can terminate EDBA as soon as s/he is happy with the result.

5 Conclusion

We studied the problem of finding a set of selection predicates on a relation that maximize the rank of a given tuple in the selection result, according to a measure attribute. The problem finds application in object promotion and characterization, however, as we show, it is NP-hard. We proposed greedy methods that find an approximate solution fast, by exploring the most promising part of the search space. Our experiments confirm the usability and efficiency of our methods on three real datasets. In the future, we plan to study the evaluation of SMP on databases with incremental updates and attributes with partially-ordered domains. We also plan to investigate the merging of multiple solutions by EDBA in a disjunction of predicate-sets that could improve the ranking of a given tuple even more (i.e., a definition of SMP that allows disjunctions). Moreover, the dual problem of SMP, which maximizes the group population while satisfying a user-specified rank threshold, is worth investigating. Although our algorithms and heuristics can be extended to solve this variant, alternative heuristics are also applicable.

References

[1] A. Arvanitis, A. Deligiannakis, and Y. Vassiliou. Efficient influence-based processing of market research queries. In *CIKM*, pages 1193–1202, 2012.
[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[3] W. W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.
[4] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD*, pages 395–406, 2006.
[5] M. Das, S. Amer-Yahia, G. Das, and C. Yu. MRI: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11):1063–1074, 2011.
[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
[7] T. Lappas, G. Valkanas, and D. Gunopulos. Efficient and domain-invariant competitor mining. In *KDD*, pages 408–416, 2012.
[8] 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.
[9] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, pages 862–873, 2009.
[10] A. Sultana, N. Hassan, C. Li, J. Yang, and C. Yu. Incremental discovery of prominent situational facts. In *ICDE*, pages 112–123, 2014.
[11] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, pages 535–548, 2009.
[12] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.
[13] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1):364–372, 2010.
[14] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):109–120, 2013.
[15] T. Wu, Y. Sun, C. Li, and J. Han. Region-based online promotion analysis. In *EDBT*, pages 63–74, 2010.
[16] T. Wu, D. Xin, Q. Mei, and J. Han. Promotion analysis in multi-dimensional space. *PVLDB*, 2(1):109–120, 2009.
[17] Y. Zhang, Y. Jia, and W. Jin. Promotional subspace mining with EProbe framework. In *CIKM*, pages 2185–2188, 2011.