Efficient Influence-Based Processing of Market Research Queries

Anastasios Arvanitis*
National Technical University of Athens, Greece
anarv@dblab.ntua.gr

Antonios Deligiannakis
Technical University of Crete, Greece
adeli@softnet.tuc.gr

Yannis Vassiliou
National Technical University of Athens, Greece
yv@cs.ntua.gr

ABSTRACT

The rapid growth of social web has contributed vast amounts of user preference data. Analyzing this data and its relationships with products could have several practical applications, such as personalized advertising, market segmentation, product feature promotion etc. In this work we develop novel algorithms for efficiently processing two important classes of queries involving user preferences, i.e. potential customers identification and product positioning. With regards to the first problem, we formulate product attractiveness based on the notion of reverse skyline queries. We then present a new algorithm, termed as RSA, that significantly reduces the I/O cost, as well as the computation cost, when compared to the state-of-the-art reverse skyline algorithm, while at the same time being able to quickly report the first results. Several real-world applications require processing of a large number of queries, in order to identify the product characteristics that maximize the number of potential customers. Motivated by this problem, we also develop a batched extension of our RSA algorithm that significantly improves upon processing multiple queries individually, by grouping contiguous candidates, exploiting I/O commonalities and enabling shared processing. Our experimental study using both real and synthetic data sets demonstrates the superiority of our proposed algorithms for the studied classes of queries.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

Keywords

reverse skylines, preferences, market research

1. INTRODUCTION

Analyzing user data (e.g., query logs, purchases) has seen considerable attention due to its importance in providing insights regarding users' intentions and helping enterprises in the process of decision making. Recently, the rapidly growing social web has been a source of vast amounts of data concerning user preferences in the form of ratings, shares, likes, etc. Previous efforts (e.g., in preference learning and recommender systems) mainly focus on helping users discover the most interesting, according to their preferences, among a pool of available products.

Highlighting the manufacturer's perspective, the need for tools to analyze user preferences for improving business decisions has been well recognized. Preference analysis has various important applications such as personalized advertising, market segmentation, product positioning etc. For example, a laptop manufacturer might be interested in finding those users that would be more interested in purchasing a laptop model. Thereby, manufacturers can benefit by targeting their advertising strategy to those users. Or they might search for laptop feature configurations that are the most popular among customers. Similarly, a mobile carrier operator that is about to launch a new set of phone plans may want to discover those plans that would collectively attract the largest number of subscribers.

In this work we develop novel algorithms for two classes of queries involving customer preferences, with practical applications in market research. In the first query type that we consider, we seek to identify customers that may find a product as attractive. We formulate this problem as a *bichromatic reverse skyline query*, and we present a new algorithm, termed as RSA, that outperforms the state-of-the-art algorithm BRS [25] in terms of both I/O and computational cost. Compared to BRS, our RSA algorithm is based on a different processing order, which allows for significant improvements with respect to the *performance*, the *scalability* and the *progressiveness* of returned results when compared to BRS.

Real world applications usually require processing multiple queries efficiently. For example, assume that a mobile carrier operator maintains a database of existing phone plans, customer statistics (i.e., voice usage duration, number of text messages sent, data volume consumed per month) and a list of new phone plans under consideration. We formulate this problem as a new query type, namely the k-Most Attractive Candidates (k-MAC) queries. Given a set of existing product specifications P, a set of customer preferences C and a set of new candidate products Q, the k-MAC query returns the set of k candidate products from Q that jointly maximizes the total number of expected buyers, measured as the cardinality of the union of individual reverse skyline sets (i.e., *influence sets*).

Recent works [12, 15] have independently studied similar problems over 'objective' attributes, i.e. those that have a globally preferred value, such as (zero) price, (infinite) battery life, etc. In such a scenario, the dominance relationships among customer preferences, existing products and candidates can be extracted by executing a single skyline query over the data set. Thereby, these works focus on providing greedy algorithms that determine the most prof-

*Anastasios Arvanitis is currently affiliated with the University of California, Riverside, USA.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

itable solution by combining customer sets. In this work we generalize their definition of user preferences, such that we can also handle 'subjective' attributes i.e., those not having a strict order for all users, e.g., as screen size, processor type, operating system etc. For example, for customer A that prefers a portable laptop, a 11" laptop would be more preferable than a 15" one. On the other hand, for a customer B searching for a desktop replacement laptop, the latter model would be more appropriate.

For such attributes, applying methods such as those proposed in [12, 15] requires having extracted the product dominance relationships for all users, since these relations are user-dependent. Thus, we have to execute a dynamic skyline query [14] for each customer, which is prohibitively expensive. Further, applying single point reverse skyline approaches to solve a k-MAC query would require calculating the influence set for each candidate product individually, which is also a very expensive task, especially when handling large data sets. We, thus, propose a batched extension of our RSA algorithm for the k-MAC problem that improves upon processing candidates sequentially by grouping contiguous candidates, exploiting I/O commonalities and enabling shared processing. After extracting the influence set of each candidate product, we also propose an algorithm that greedily calculates the final solution for the k-MAC problem by combining the influence sets of individual candidate products. In brief, the contributions of this paper are:

- We present a novel progressive algorithm, termed as RSA, for (single) reverse skyline query evaluation. Our RSA algorithm scales better in data sets that contain a large number of skyline points (e.g., high-dimensional data), while reporting the first results significantly faster than the state-of-the-art algorithm BRS.
- We develop a batched variant of our RSA algorithm that improves upon processing multiple queries individually, by grouping contiguous candidates, exploiting I/O commonalities and enabling shared processing among similar candidates. We then apply our batched algorithm to solve the k-MAC query. k-MAC generalizes the "k-most demanding products query" of [12] and the "top-k popular products query" of [15] to problems where customer preferences also include subjective attributes.
- We perform an extensive experimental study using both synthetic and real data sets. Our study demonstrates that (*i*) our RSA algorithm outperforms BRS for the reverse skyline query, in terms of I/Os, CPU cost and progressiveness of the output, especially for real data, higher dimensional data, or when the size of the product data set is relatively larger than that of customers, and (*ii*) that our proposed batched algorithm outperforms baseline approaches that process each candidate individually.

2. PRELIMINARIES

2.1 Single Point Reverse Skylines

Consider two sets of points, denoted as P and C, in the same D-dimensional space. We will refer to each point $p \in P$ as a *product*. Each product is a multi-dimensional point, with p_i denoting the product's attribute value A_i. For example, assuming that products are notebooks, the dimensions[1] of p_i may correspond to the notebook's price, weight, screen size, etc. Further, each point $c \in C$ represents a customer's preferred notebook specifications that she would be interested in; we will refer to each point c as a *customer*. Clearly, customers are more interested to the products that are closer to their preferences. In order to capture the preferences of a customer c, we formally define the notion of dynamic dominance.

DEFINITION 1. *(Dynamic Dominance) (from [5]): Let $c \in C$, $p, p' \in P$. A product p dynamically dominates p' with respect to c, denoted as $p \prec_c p'$, iff for each dimension $|p_i - c_i| \leq |p'_i - c_i|$ and there exists at least one dimension such that $|p_i - c_i| < |p'_i - c_i|$.*

Note that this definition can accommodate dimensions with universally optimal values where smaller (larger) values are preferred by simply setting c_i to the minimum (resp. maximum) value of dimension A_i. For example, assuming that lighter notebooks are preferred, we can simply set for all customers $c_{weight} = 0$.

DEFINITION 2. *(Dynamic Skyline) (from [5]): The dynamic skyline with respect to a customer $c \in C$, denoted as $SKY(c)$, contains all products $p \in P$ that are not dynamically dominated with respect to c by any other $p' \in P$.*

Consider a set of existing products $P = \{p_1, p_2, p_3, p_4\}$ and customers $C = \{c_1, c_2, c_3\}$. Figure 1(a) illustrates the dynamic skyline of c_1 that includes notebooks p_2 and p_4 in a sample scenario with 2 dimensions corresponding to the CPU speed and the screen size of a notebook. Points in the shaded areas are dynamically dominated by points belonging to the dynamic skyline of c_1. Since we are interested in the absolute distance between products, a product might dominate other products that belong to different quadrants with respect to a customer. For example, p_1 and p_3 in the upper right quadrant are dynamically dominated by p_2 in the lower right quadrant because p_2 has a CPU speed and a screen size that are both closer to c_1 than the corresponding characteristics of p_1 and of p_3. Figures 1(b) and 1(c) illustrate the dynamic skylines of customers c_2 and c_3 respectively. We now highlight the product's perspective by introducing the definition of *bichromatic reverse skylines*.

DEFINITION 3. *(Bichromatic Reverse Skyline) (from [11]): Let P be a set of products and C be a set of customers. The bichromatic reverse skyline of p, denoted as $RSKY(p)$ contains all customers $c \in C$ such that $p \in SKY(c)$.*

Thus, the bichromatic reverse skyline of a product p contains all customers c that find p as 'attractive'. Henceforth, we refer to the bichromatic reverse skyline of p as the *influence set* of p. Figure 1(d) illustrates the influence sets of products p_1, p_2, p_3 and p_4. The cardinality of $RSKY(p)$ is a useful metric of the product's impact in the market. We refer to $|RSKY(p)|$ as the *influence score* $\mathrm{IS}(p)$. In our example, $\mathrm{IS}(p_1)=\mathrm{IS}(p_2)=2$ and $\mathrm{IS}(p_3)=\mathrm{IS}(p_4)=1$.

2.2 Influence Region

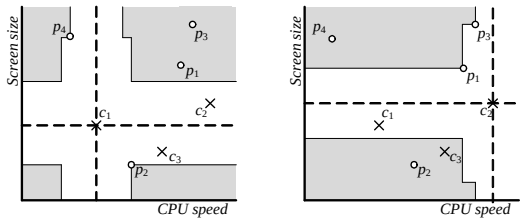
Consider a new product q. The new product partitions the D-dimensional space into 2^D orthants Ω_i, each identified by a number in the range $[0, 2^D - 1]$. Since all orthants are symmetric and we are interested in the absolute distance between products, we can map all products to Ω_0 as illustrated in Figure 2(a). For simplicity, we hereafter concentrate on Ω_0 with respect to a query point q.

For every dynamic skyline point p_i, let $m_i(q)$ be the midpoint of the segment connecting a query point q with p_i. In Figure 2(b) black points m_1, m_2 and m_4 represent the midpoints of p_1, p_2 and p_4 with respect to q. Henceforth, in order to alleviate the complication of maintaining both points and midpoint skylines, whenever we refer to a product p_i we imply the corresponding $m_i(q)$ with respect to q. We also assume that each dynamic skyline point p_i with respect to q is mapped to its midpoint skyline $m_i(q)$ on the fly.

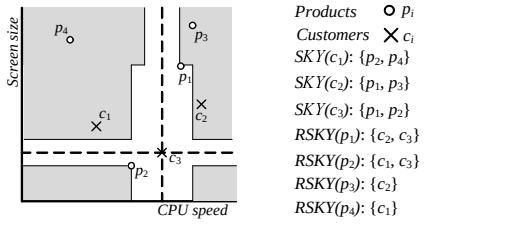
The *influence region* of a query point q, denoted as $\mathrm{IR}(q)$, is the union of all areas not dynamically dominated with respect to q by the midpoint skylines of q. The area in Ω_0 that is not shaded in Figure 2(b) draws the influence region for q. Note that the midpoints themselves belong to the IR, since a tuple cannot dominate itself.

LEMMA 1. *(from [11]) A customer c belongs to the influence set $RSKY(q)$ of q iff c lies inside the influence region of q i.e., $c \in IR(q) \Leftrightarrow c \in RSKY(q)$.*

[1] In the following we will use the terms dimension and attribute interchangeably

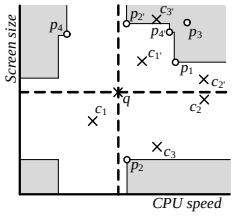


(a) Dynamic Skyline of c_1 (b) Dynamic Skyline of c_2

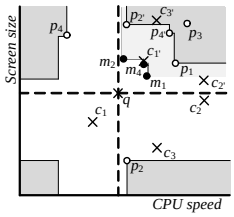


(c) Dynamic Skyline of c_3 (d) Skylines and Influence Sets

Figure 1: Dynamic Skylines example

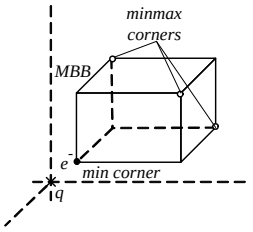


(a) Transformed space to Ω_0 with respect to q

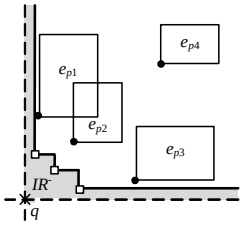


(b) Midpoints w.r.t. q

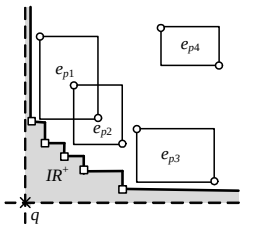
Figure 2: Influence region of q



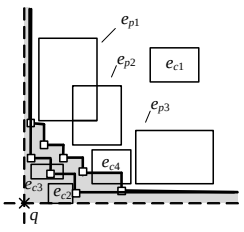
(a) Example MBB



(b) Lower bound for IR(q)



(c) Upper bound for IR(q)



(d) Pruning example

Figure 3: Influence Regions

Returning to the example of Figure 2(b), notice that only c_2 lies inside IR(q). Therefore, $RSKY(q) = \{c_2\}$.

Hereafter, we assume that all points (either products or customer preferences) are indexed using a multidimensional index (e.g., R-trees, kd-trees etc.); for our presentation we will consider R-trees. Figure 3(a) shows an example minimum bounding box (MBB) e. Inside each MBB e, let *min-corner* $e^-(q)$ denote the point in e having the minimum distance from a query point q. The min-corner dominates the largest possible space. The points that reside in each of the d faces closest to q and are the farthest from the origin q are denoted as *minmax-corners*. Each MBB contains D minmax-corners. Independently of how products within e are distributed, any point in e certainly dominates the area that the minmax-corners do, while at best it dominates the area that the min-corner does.

Given a set of MBBs, we can derive two sets: the set of all min-corners denoted as L and the set of all minmax-corners w.r.t. q denoted as U. Figure 3(b) presents an example, assuming $E_P = \{e_{p1}, e_{p2}, e_{p3}, e_{p4}\}$, where e_{p_i} denotes a product entry. In Figures 3(b), 3(c) black and hollow circles represent the min-corners and minmax-corners respectively and rectangles represent midpoints.

Continuing the example of Figure 3(b), the grey area represents a lower bound of the actual influence region $\text{IR}^-(q)$ and it is defined as the space not dominated w.r.t. q by any min-corner $l \in L$. Respectively, the grey area in Figure 3(c) represents an upper bound of the actual influence region $\text{IR}^+(q)$, defined as the space not dominated w.r.t. q by any minmax-corner $u \in U$. It follows [25]:

LEMMA 2. *If an entry e_c is dominated by any $u \in U$, i.e. e_c is completely outside $IR^+(q)$, e_c cannot contain any customer inside $IR(q)$. Hence, according to Lemma 1, e_c can be pruned.*

For example, e_{c1} in Figure 3(d) can be pruned because it is completely outside $\text{IR}^+(q)$.

2.3 The BRS Algorithm

In the following we detail the state-of-the-art *Bichromatic Reverse Skyline* (BRS) algorithm [25] that efficiently calculates the influence set of a single query point q. BRS aims at minimizing the I/O cost (*i*) by progressively refining the influence region of q until the influence set of q has been retrieved, (*ii*) by applying Lemma 2 to prune e_c entries that do not contribute to $RSKY(q)$.

BRS uses two indexes, an R-tree T_P on the set of products P and another T_C on the set of customers C. Initially, the algorithm inserts all root entries of T_P (resp. T_C) in a priority queue E_P (resp. E_C) sorted with the minimum Euclidean distance of each entry from q. BRS extracts a set L of all min-corners and a set U of all minmax-corners of $e_p \in E_P$. Further, in order to reduce the number of subsequent dominance checks, BRS calculates the skylines of L and U, denoted as $SKY(L)$ and $SKY(U)$ respectively.

In each iteration BRS expands the entry in E_P with the minimum Euclidean distance from q and updates the current L and U and their skylines $SKY(L)$ and $SKY(U)$. Then, all $e_c \in E_C$ are checked for dominance with $SKY(L)$ and $SKY(U)$. If e_c is not dominated by $SKY(L)$ (i.e. it intersects $\text{IR}^-(q)$), BRS expands e_c as it may contain customers inside IR(q). Returning to Figure 3(d), e_{c3} intersects $\text{IR}^-(q)$; therefore e_{c3} is expanded. In contrast, if a customer entry e_c (such as e_{c1} in Figure 3(d)) is dominated by $SKY(U)$, then e_c can be safely pruned according to Lemma 2. BRS terminates when E_C becomes empty, i.e. the position of all customers either inside or outside IR(q) has been determined.

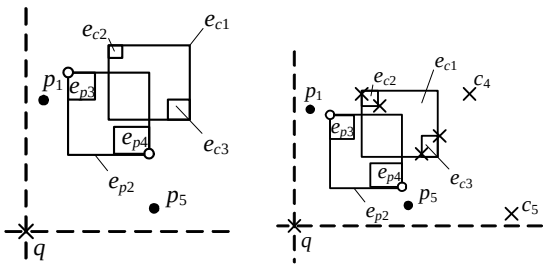
3. EFFICIENT COMPUTATION OF REVERSE SKYLINES

In this section, we detail the drawbacks of BRS and then present a more efficient reverse skyline algorithm, termed RSA.

3.1 BRS Shortcomings

Complexity Analysis. Let p_k, c_k denote the sizes of the currently active entries in E_P and E_C, respectively, after the k-th iteration of the BRS algorithm. The worst-case cardinality of p_k and c_k are $|P|$ and $|C|$ respectively. In each iteration, the BRS algorithm maintains both $SKY(L)$ and $SKY(U)$, two sets with $O(|P|)$ and $O(D|P|)$ entries respectively, where D is the dimensionality of the data set. BRS then checks for dominance each entry in E_P and E_C with both $SKY(L)$ and $SKY(U)$. Thus, each iteration entails $O(D|P| \times (|P| + |C|))$ dominance checks, which require a total of $O(D^2|P| \times (|P| + |C|))$ comparisons, since each dominance check requires $O(D)$ comparisons.

Clearly, the processing cost of BRS depends on the size of the intermediate upper and lower skyline sets. [2] shows that for uniformly distributed data the size of the skyline set is $\Theta(\frac{(ln|P|)^{D-1}}{D!})$. Thus, for larger data sets or higher dimensional data the processing cost of maintaining $SKY(L)$ and $SKY(U)$ becomes prohibitively expensive. Our experimental evaluation (Section 6) confirms that



(a) BRS will expand e_{p_2} entailing an unnecessary I/O

(b) RSA will avoid accessing e_{p_2} by pruning c_4 with p_1

Figure 4: Processing order and I/O accesses

BRS is impractical for $|P| \geq 10^6$ or $D \geq 4$. Motivated by the above analysis, we introduce a more efficient and scalable reverse skyline algorithm, which eliminates the dependency on the $SKY(L)$ and $SKY(U)$ sets, thus being able to handle high dimensional data, or, in general, data where the size of skyline points is large.

Processing Order. BRS performs a synchronous traversal on the T_P and T_C indexes, which are built on product and customer points, respectively, following a monotonic order based on the Euclidean distance of e_p entries from q. This processing order ensures that the number of I/Os on T_P is minimized. However, in terms of the total I/Os, BRS might perform some unnecessary I/Os. Figure 4(a) illustrates one such scenario, where the nodes e_{p_2} and e_{c_1} have not been yet expanded.[2] BRS would proceed by expanding e_{p_2}, revealing e_{p_3} and e_{p_4}. Unfortunately, e_{c_1} is not affected by this refinement and it still has to be accessed. On the other hand, if we first expand e_{c_1}, this operation would reveal e_{c_2} and e_{c_3}, which can be pruned by p_1 and p_5 respectively, eliminating the need to access e_{p_2}. Clearly, in this scenario the I/O access on e_{p_2} was redundant. In order to avoid such redundant I/Os, our RSA follows a visiting strategy that is primarily based on the *tree level* of customer entries, which, as confirmed in our experiments, results in fewer total I/Os.

Progressiveness. BRS iteratively refines $\text{IR}^-(q)$ and reports the customer points that lie inside $\text{IR}(q)$. In order to retrieve the first reverse skyline point, several iterations of BRS may be required, which is undesirable for applications that require a quick response containing only a fraction of the output, or if the complete output is not useful (e.g., if it contains too many results). We, thus, seek to develop an algorithm that reports the first results faster than BRS.

3.2 The RSA Algorithm

We now present our *Reverse Skyline Algorithm* (RSA), which aims to address the shortcomings outlined above.

Data Structures Used and Basic Intuition. The RSA algorithm:

- Does not require the maintenance of the $SKY(L)$ and $SKY(U)$ sets and is, thus, less expensive in terms of processing cost.
- Checks one customer entry per iteration following a visiting strategy based on the entry's tree level (primary sort criterion) and Euclidean distance from q (secondary sort criterion).
- Accesses a product entry only if it is absolutely necessary in order to determine if a customer point belongs to $RSKY(q)$.

RSA maintains the following data structures for its operation:

- A priority queue E_P on the set of product entries
- A priority queue E_C on the set of customer entries
- A set $SKY(q)$ with the currently found midpoint skylines

The two priority queues are sorted based on a dual sorting criterion: primarily, based on the tree level of the stored entries and, subsequently, using the Euclidean distance of each entry from q.

[2] Note that with e_p we actually represent the respective midpoints w.r.t q.

Algorithm 1: RSA

Input: q a query point, T_P R-tree on products, T_C R-tree on customers, $E_P(q)$ priority queue on products, $E_C(q)$ priority queue on customers
Output: $RSKY(q)$ reverse skylines of q
Variables: $SKY(q)$ currently found midpoint skylines of products w.r.t. q

```
1  begin
2      SKY(q) := ∅; RSKY(q) := ∅;
3      while E_C ≠ ∅ do
4          dominated := false;
5          E_C(q).pop() → e_c;
6          if dominated(e_c, SKY(q)) then
7              dominated := true; continue;
8          if e_c is a non-leaf entry then
9              Expand e_c, insert children entries in E_C(q);
10         else
11             foreach e_p ∈ E_P(q) do
12                 midpoint(e_p, q) → m;
13                 if e_c is dominated by m then
14                     if e_p is a leaf entry then
15                         if (dominated(m, SKY(q)) == false) then
16                             SKY(q).push(m);
17                         dominated := true; break;
18                     else
19                         Expand e_p, insert children entries in E_P(q);
20                     E_P(q).remove(e_p);
21             if (dominated == false) then
22                 RSKY(q).push(e_c);
23     return RSKY(q);
```

Thus, leaf entries are given higher priority and are processed first, while the examination of non-leaf entries is postponed as much as possible. By first processing all leaf e_c entries, the algorithm may reveal a midpoint skyline, which will be subsequently used to prune a non-leaf e_c based on Lemma 2, thus avoiding an access on T_C. The same intuition holds for e_p entries as well; an already found midpoint skyline can be used to prune dominated non-leaf product entries, since these entries will not contribute to the skyline. Further, whenever an e_c entry is checked for dominance with E_P, first all leaf e_p entries will be examined. As long as no leaf e_p dominates e_c, only then will RSA proceed to expand the nearest to q non-leaf e_p entry. This change in the visiting order of E_P reduces the number of accesses on T_P as well. For instance, in Figure 4(b) BRS would access e_{p_2} that has the minimum Euclidean distance from q. In contrast, RSA will use p_1 to determine that c_4 does not belong to $RSKY(q)$, hence avoiding the access of e_{p_2}.

Algorithm Description. The RSA algorithm is presented in Algorithm 1. Initially, RSA inserts all root entries of T_P (resp. T_C) in the priority queue E_P (resp. E_C). Further, RSA maintains a set $SKY(q)$ of the currently found midpoint skylines which are used for pruning based on Lemma 1. RSA proceeds in iterations. In each iteration RSA extracts the entry in E_C having the minimum key from q (Line 5) and checks the following pruning conditions:

1. If e_c is dominated by any point that belongs to the currently found midpoint skylines $SKY(q)$, e_c can be removed from E_C based on Lemma 1 (Lines 6-7).
2. Otherwise, if e_c is a non-leaf entry (Line 8), e_c is expanded and child nodes are inserted into E_C (Line 9).
3. Else, for all e_p entries in E_P (Lines 11-22):
   - If e_c is dominated by the midpoint of a leaf entry $e_p \in E_P$ (Line 13), then e_c can be removed from E_C, based on Lemma 1, and the midpoint of e_p is inserted into $SKY(q)$ (Line 16)).
   - Else if e_c is dominated by the midpoint of the *min-corner* e_p^- of a non-leaf $e_p \in E_P$ (Line 18), e_p is expanded and its children entries are inserted into E_P (Line 19).

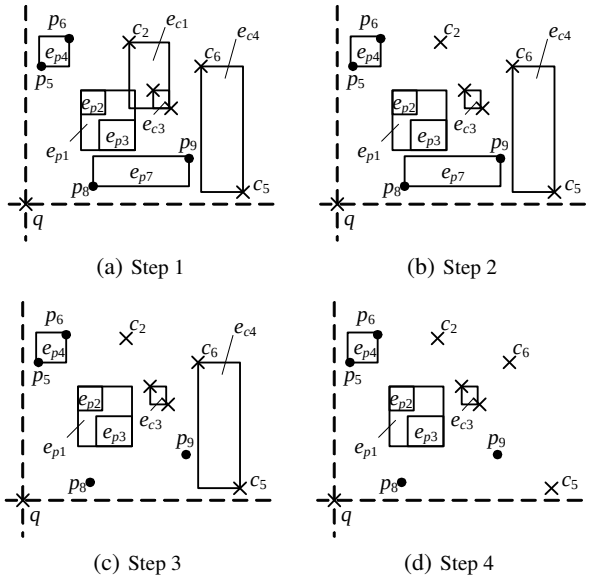


Figure 5: Running example of the RSA algorithm

Finally, if e_c has not been pruned by any of the above conditions (Line 21), then e_c is a reverse skyline point and can be at that stage reported as a result (Line 22). The RSA algorithm terminates when E_C becomes empty and then $RSKY(q)$ is returned (Line 23).

Example. We illustrate the execution of RSA using the running example depicted in Figure 5. At the beginning, $E_P(q) = \{e_{p_7}, e_{p_1}, e_{p_4}\}$ and $E_C(q) = \{e_{c_1}, e_{c_4}\}$ (sorted by their distance from q). In the first iteration, RSA will examine e_{c_1} that has the minimum distance from q. Since it is a non-leaf entry, RSA will expand e_{c_1} (Line 9) and it will insert child nodes c_2 and e_{c_3} into $E_C(q)$ (see Figure 5(b)). Now $E_C(q) = \{c_2, e_{c_4}, e_{c_3}\}$ and RSA selects to examine c_2. Since the current skyline is empty, c_2 is not dominated by any product entry; hence RSA will proceed by checking if c_2 is dominated by any product entry contained in $E_P(q)$. c_2 is dominated by the min-corner of the first entry in $E_P(q)$, i.e. e_{p_7} (Line 13). In order to determine if there actually exists a point inside e_{p_7} that dominates c_2 w.r.t. q, RSA will expand e_{p_7} (Line 19), pushing its child nodes p_8 and p_9 inside $E_P(q)$, thus $E_P(q) = \{p_8, p_9, e_{p_1}, e_{p_4}\}$ (see Figure 5(c)). Now, RSA discovers that c_2 is dominated by p_8, which is marked as a skyline point (Line 16) and c_2 is discarded. In the next iteration, RSA selects to examine e_{c_4}, which has the minimum distance from q. e_{c_4} is not dominated by any currently found skyline point and since it is a non-leaf entry, it is expanded and child nodes c_5 and c_6 are inserted into $E_C(q)$ (Line 9) (see Figure 5(d)). Now we have $E_C(q) = \{c_5, c_6, e_{c_3}\}$. Next, RSA will examine c_5. Since c_5 is not dominated by any product entry, c_5 is reported as a reverse skyline result (Line 22). Now RSA examines c_6 which is already dominated by a currently found skyline point, i.e. p_8, (Line 6), hence it is discarded. Finally, e_{c_3} is examined. Similarly, e_{c_3} is dominated by p_8 and it is also pruned. Since $E_C(q)$ is now empty, RSA terminates and outputs c_5 as the final answer.

Complexity and Progressiveness Analysis. RSA requires at most $|C|$ iterations (one for each customer), although in fact several e_c entries will be pruned by $SKY(q)$ (Line 6). Each iteration entails a dominance check with (i) the currently found midpoint skyline $SKY(q)$, and (ii) all product entries currently in E_P, both having $O(|P|)$ worst-case cardinality. Overall RSA requires $O(|P||C|)$ dominance checks, or $O(D|P||C|)$ comparisons.

With respect to progressiveness, recall that RSA will first examine leaf customer entries that have the minimum Euclidean distance from the query point q (based on the dual sorting scheme on E_C). In other words, the very first iterations of RSA concern customer entries that are very close to q. Intuitively, the closer to q a customer is, the more likely that q will not be dominated by any product w.r.t. the examined customer. Hence, customers that will be examined in the first iterations tend to have a higher probability of belonging to $RSKY(q)$. Further, since the first entries to be examined are actual points (not MBBs), the first iterations will not involve e_c expansions (Line 10), which are expensive in terms of processing cost. Thus, the first iterations will be faster than subsequent ones. Overall, RSA typically reports the first results in just a few iterations. In contrast, recall that BRS requires several iterations in order to adequately refine the influence regions, such that the first reverse skylines have been determined. Our experimental study (Section 6) verifies the superiority of RSA in terms of progressiveness compared to the BRS algorithm.

4. K-MAC QUERIES

We now present the k-Most Attractive Candidates (k-MAC) query, which serves as a motivating example that demonstrates the need to develop a batch processing algorithm for computing several reverse skyline queries. The k-MAC query is a slight generalization of the problems studied in [15, 12] for the case when the customer preferences also include subjective dimensions. We first present a motivating scenario, which highlights the usefulness of k-MAC queries. We then present the definition of the k-MAC query.

Motivating Scenario. A laptop manufacturer wants to produce k new notebooks, among a set of feasible alternative configurations Q proposed by the engineering department. The manufacturer needs to consider three sets: (i) the existing competitor products P, (ii) the set of customers' preferred specifications C, and (iii) a set of candidate products Q. We will refer to each $q \in Q$ as a *candidate*. The goal of the manufacturer is to identify the specifications that are expected to *jointly* attract the largest number of potential buyers. Note that this is different than simply selecting the k products that are the most attractive individually, since it does not make much sense to select products that seem attractive to the same set of customers.

Problem Definition. We first define the *joint influence set* for a set of candidates Q. We then define the notion of the *joint influence score* and introduce the *k-Most Attractive Candidates (k-MAC)* query.

DEFINITION 4. *(Joint Influence Set): Given a set of products P, a set of customers C and a set of candidates Q, the joint influence set of Q, denoted as $RSKY(Q)$, is defined as the union of individual influence sets of any $q_i \in Q$: $RSKY(Q) = \bigcup_{q_i \in Q} RSKY(q_i)$*

Following the above definition, the *joint influence score* $\mathrm{IS}(Q)$ for a set of candidates Q is equal to the size of the joint influence set of Q, $|RSKY(Q)|$. We now introduce the *k-Most Attractive Candidates* (k-MAC) query as follows:

DEFINITION 5. *(k-Most Attractive Candidates (k-MAC) query): Given a set of products P, a set of customers C, a set of candidates Q and an integer $k > 1$, determine the subset $Q' \subseteq Q$, such that $|Q'| = k$ and the joint influence score of Q', $IS(Q')$, is maximized.*

Note that several candidates might be interesting for the same customer. Additionally, we emphasize that for evaluating a k-MAC query each candidate $q \in Q$ is considered separately from other candidates and only with respect to existing products. In other words, intra-candidate dominance relations are not taken into account for k-MAC queries. This is consistent with a real-world setting where a manufacturer is interested to compare their product portfolio only with respect to the competition. We discuss how we resolve ties at the end of this section where we present a greedy algorithm that computes an approximate solution for the k-MAC problem.

Unlike recent works [15, 12] that targeted similar problems assuming only 'objective' attributes, i.e. those having a globally preferred value (such as zero price, infinite battery life, etc.), k-MAC can handle cases where customer preferences are expressed over 'subjective' dimensions (e.g., screen size, processor type). This generalisation is possible because the attractiveness of each candidate products is computed based on the size of their bichromatic influence set. Moreover, while our focus is on efficiently computing the influence sets of multiple candidate products, the emphasis of [15, 12] is on the selection of the proper candidates *after* the dominance relationships among products have been determined.

A Greedy Algorithm. Unfortunately, processing k-MAC queries is non-trivial. This problem can be reduced to the more general *maximum k-coverage* problem. Thus, even if we consider the much simpler problem where all the influence sets of all candidates have been computed, an exhaustive search over all possible k-cardinality subsets of Q is NP-hard. Based on the complexity of computing the subset of k products, we now seek an efficient, greedy algorithm for this problem. Our solution is based on the generic k-stage covering algorithm provided in [8], developed for finding efficient approximate solutions to the maximum k-coverage problem.

LEMMA 3. *(from [8]) k-stage covering algorithm returns an approximate solution to the maximum k-coverage problem that is guaranteed to be within a factor* $1-1/e$ *from the optimal solution.*

We now show how we can adapt the k-stage covering algorithm for the k-MAC problem. kGSA (k-stage Greedy Selection Algorithm) takes as input a set of candidate products Q and their associated influence sets and returns a set $Q' \subseteq Q$, $|Q'| = k$ that contains the candidates which formulate a $(1-1/e)$-approximate solution to the k-MAC query. kGSA proceeds in iterations, by adding one candidate into Q' during each iteration. All candidates are examined at each iteration, and kGSA selects the one that, if added in Q', results in the largest increase of the joint influence score of Q'. In case multiple candidates contribute equally to the increase of $\mathrm{IS}(Q')$, kGSA applies a second criterion; it selects the candidate with the minimum sum of distances from its respective reverse skylines (customers). The intuition is that a candidate product that is closer to a user's preferences would more likely increase user satisfaction. kGSA terminates after k iterations and returns Q'.

5. REVERSE SKYLINE PROCESSING FOR MULTIPLE QUERY POINTS

Solving the k-MAC problem requires processing all candidates, in order to first determine their influence sets. The kGSA algorithm can then be used to solve the k-MAC problem.

A straightforward way to process multiple candidates would be to apply either BRS or RSA for each candidate individually. However this approach is very inefficient in terms of I/O accesses, because it requires accessing each entry e_p (e_c) several times, if e_p (e_c) appears in the priority queues of more than one candidate.

Our bRSA algorithm. We now introduce our bRSA algorithm, which aims at eliminating the drawbacks of the baseline approach by exploiting I/O commonalities and by offering shared processing among candidates. bRSA utilizes in its core the RSA algorithm that we presented in Section 3. Note that, apart from k-MAC queries, bRSA can be applied for any query type that requires joint processing of multiple reverse skyline queries.

bRSA efficiently processes candidates in parallel, by grouping them in batches, in such a way that grouped candidates benefit by the processing of other group members. A primary goal of bRSA is to save duplicate I/O accesses, by using entries that have been expanded during an iteration of the RSA subcomponent for one group member, in order to prune entries that appear in the local priority queues of other group members as well. In particular, whenever an entry e_x is expanded, all local priority queues in which e_x appears are appropriately updated. Hence, each disk page is accessed only once per batch. Additionally, in order to further optimize the processing of group members, bRSA maintains a list of currently found product points, that will expectedly have large pruning potential for other group members, based on Lemma 1. We will refer to these points as *vantage points* and we will explain their use in the following where we discuss bRSA execution in detail.

Note that we cannot safely assume that all the necessary data structures that bRSA utilizes (local priority queues, skyline sets for each candidate, list of vantage points etc.) will actually fit in main memory. Based on the memory capabilities of the hardware and worst case estimates of the amount of customer and product entries in E_P and E_C, let us assume that G candidates (where $G \ll |Q|$) fit in main memory and can be simultaneously processed. Using worst case estimates does not have a severe impact in the performance of bRSA; in fact, as we demonstrate experimentally, it is better to keep G to fairly modest values (i.e., up to 10 candidates). Larger batch sizes may result in increasing processing cost for the maintenance of local priority queues and significantly more dominance checks which gradually eliminates the benefit from shared processing.

Candidates in proximity in the multidimensional space are more likely to benefit from shared processing. Hence, as a preprocessing step, bRSA partitions the candidate set into $\lceil |Q|/G \rceil$ batches based on a locality preserving hashing method, such as the Hilbert space filling curve.[3] Then, bRSA picks one candidate at a time in a round robin fashion (Line 5), and executes a single iteration of a modified version of the RSA algorithm for that candidate, termed Batch-RSA. Batch-RSA extends RSA to be efficiently used on a batch setting. We now present the differences of Batch-RSA compared to its single point counterpart. First, whenever an entry e_x is expanded, all local priority queues in which e_x appears are appropriately updated. Further, when a leaf product entry, say p_i, is discovered (Line 12), the algorithm decides whether p_i should be inserted to a buffer H_P that contains vantage points, i.e. those that will be used for pruning by other candidates (Line 17). Intuitively, product points that reside closer to a candidate, will dominate the largest possible space and their pruning power will be maximized. Thus, we implemented H_P as a priority queue on the minimum Euclidean distance, among the candidates inside the batch. If H_P is full, the most distant point in H_P, is replaced with p_i. Vantage points (essentially their respective midpoints) are used additionally to skyline points when checking each customer entry for dominance (second condition in OR clause of Line 5), hence avoiding some of the subsequent I/Os.

6. EXPERIMENTAL EVALUATION

All algorithms examined in our experiments were implemented in C++ and executed on a 2.0 GHz Intel Xeon CPU with 4 GB RAM running Debian Linux. The code for the BRS algorithm was thankfully provided to us by the authors of [25].

6.1 Experimental Setup

We used a publicly available generator [1] in order to construct different classes of synthetic data sets, based on the distribution of the attributes' values; i.e., uniform (UN), anti-correlated (AC) and correlated (CO). Due to space limitations, in the following we plot the results primarily for uniform (UN) data sets. Experiments involving AC and CO data, as well as combinations among them (e.g., uniformly distributed products and anti-correlated customers)

[3] Other clustering techniques might also be applicable

Algorithm 2: bRSA

Input: Q a set of candidates, T_P R-tree on products, T_C R-tree on customers
Variables: $E_P(q_i)$ priority queue on products for q_i, $E_C(q_i)$ priority queue on customers for q_i, $RSKY(q_i)$ reverse skylines for q_i, $SKY(q_i)$ midpoint skylines of q_i, G_j batches with $|G_j| = G$

```
1  begin
2      partition Q into ⌈|Q|/G⌉ batches → G_j;
3      foreach G_j do
4          while (RSKY(q_i) for all q_i ∈ G_j have not been found) do
5              selectCandidate → q_i;
               /* Process q_i until IS(q_i) has been
                  completely determined                    */
6              if E_C(q_i) ≠ ∅ then
7                  Batch-RSA (q_i, G_j, T_P, T_C, E_P(q_i), E_C(q_i), RSKY(q_i),
                   SKY(q_i), H_P);
```

Function Batch-RSA

Input: G a group of candidates, T_P R-tree on products, T_C R-tree on customers, $E_P(q_i)$ priority queue on products for q_i, $E_C(q_i)$ priority queue on customers for q_i, $RSKY(q_i)$ reverse skylines of q_i, $SKY(q_i)$ midpoint skylines of q_i, H_P priority queue on product leaf entries (vantage points)
Output: $RSKY(q_i)$ reverse skylines of q_i

```
1  begin
2      while E_C(q_i) ≠ ∅ do
3          dominated := false;
4          E_C(q_i).pop() → e_c;
5          if dominated(e_c, SKY(q_i)) OR dominated(e_c, H_P) then
6              dominated := true; continue;
7          if e_c is a non-leaf entry then
8              Expand e_c for all relevant q_i, insert children into E_C(q_i);
9          else
10             foreach e_p ∈ E_P(q_i) do
11                 midpoint(e_p, q_i) → m;
12                 if e_c is dominated by m then
13                     if e_p is a leaf entry then
14                         if (dominated(m, SKY(q_i)) == false) then
15                             SKY(q_i).push(m);
16                         H_P.push(e_p);
17                         dominated := true; break;
18                     else
19                         Expand e_p for all relevant q_i, insert children into
                           E_P(q_i);
20                     E_P(q_i).remove(e_p);
21             if (dominated == false) then
22                 RSKY(q_i).push(e_c);
23     return RSKY(q_i);
```

generally follow similar trends. We also evaluated our algorithms on two real world data sets. The NBA data set (NBA) consists of 17,265 5-dimensional points, representing the average values of a player's annual performance with respect to the number of points scored, rebounds, assists, steals and blocks. The household data set (HOUSE), consists of 127,930 6-dimensional points, representing the percentage of an American family's annual income spent on 6 types of expenditure: gas, electricity, water, heating, insurance, and property tax. In order to generate customer and candidate sets, we added Gaussian noise to actual points. For both synthetic and real data sets we normalized all attribute values to [0,10000] and for each data set, we built an R-tree with a page size equal to 4KB.

We compared the performance of our RSA and bRSA algorithms with the state-of-the-art BRS algorithm for evaluating both reverse skyline and k-MAC queries. For BRS and RSA, we measured the total CPU time and I/O operations required for processing *(i)* a workload of $|Q|$ reverse skyline queries, and *(ii)* a k-MAC query given an input of $|Q|$ candidate products. The bRSA algorithm applies only for the k-MAC query. In particular, we measured:

- The number of I/Os (separately on product and customer entries). For each data set, one memory buffer equivalent to 100 pages (12.5% of the data set size) was allocated for caching, following a Least Recently Used (LRU) cache replacement policy.
- The time spent on CPU.
- The total query processing time, consisting of the time spent on CPU plus the I/O cost, where each random page access was penalized with 1 millisecond.

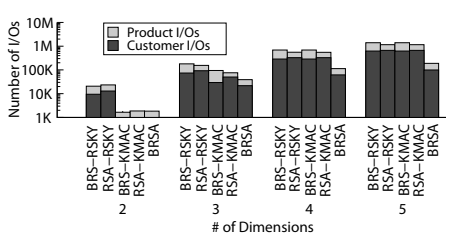
Recall that for the reverse skyline query type, both BRS and RSA process query points sequentially. For evaluating k-MAC queries, we modified both algorithms by adding *(i)* a preprocessing step that presorts candidates based on their Hilbert hash value, and *(ii)* a final step that greedily outputs the best candidates using our kGSA algorithm. In our experiments the measured processing time required for both steps was negligible compared to the time required for the algorithm execution. Further, it is important to emphasize that none of the algorithms is affected by the value of k, since they first have to determine the influence sets of all candidates, and then greedily select the optimal k-subset based on kGSA.

In each experiment we vary a single parameter while setting the remaining to their default values. The default values for both product and customer data set cardinalities were set to 100,000, the default data dimensionality was set to 3, the default domain range of each attribute was [0, 10000], the default batch size was set to 10, and the default buffer size was set to 12.5% of the data set size.

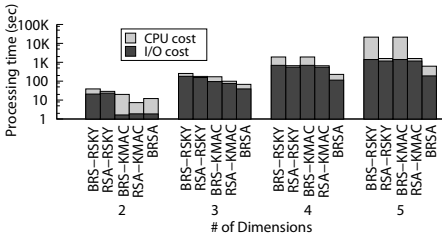
6.2 Experimental Results

Sensitivity Analysis vs. Data Dimensionality. We first vary the dimensionality of the data sets from 2 to 5 and examine the performance of all algorithms. Figures 6(a)-6(b) show the results for the number of I/Os and the total processing time, respectively, in logarithmic scale. The corresponding numbers for the BRS and RSA algorithms are also presented, for clarity, in Figure 6(c). As expected (refer to the complexity analysis in Section 3), BRS becomes prohibitively expensive for data with more than 3 dimensions. In particular, BRS requires 3.35 times more CPU time than RSA even in 2 dimensions, and is about 46 times slower in terms of CPU time, and 13.5 times slower in terms of total processing time in the 5-dimensional data set. Figure 6(k) shows an analogous behaviour of the algorithms in anti-correlated data. We also experimented with higher dimensionalities, e.g., for $D = 6$, BRS took $\sim$15 hours to finish, whereas RSA terminated in 20.2 minutes. However, we did not include these results in the plots due to space limitations. Our experiments with real data sets show that BRS is impractical for higher dimensions, which justifies our motivation for a more efficient reverse skyline algorithm. It is important to notice that in higher dimensionalities the I/O cost of BRS is dominated by the CPU cost (note that Figures 6(a)-6(b) are in logarithmic scale). To understand why BRS escalates poorly with D recall that the sizes of $SKY(L)$ and $SKY(U)$, which are maintained by BRS, increase rapidly with the data dimensionality [2]. Finally, our bRSA algorithm achieves significant performance gains with respect to both BRS and RSA in all settings. Note that our remaining sensitivity analysis using synthetic data sets, utilizes a modest value ($D = 3$), which is a favorable setting for BRS. Obviously, the benefits of our algorithms over BRS were significantly more in higher dimensions.

Sensitivity Analysis vs. Data Set Size. We then perform a sensitivity analysis with respect to the size of the product data set. Notice the different behavior of the two algorithms with respect to the type of I/Os (Figure 6(d)), due to the different visiting orders followed; generally BRS entails more accesses on the products index, whereas RSA requires more customer I/Os. In terms of I/Os, RSA exhibits similar performance with BRS in the case where the product and customer data have the same size (100K both). However, as the number of products increases, the strategy followed



(a) I/Os vs. Dimensionality (UN)

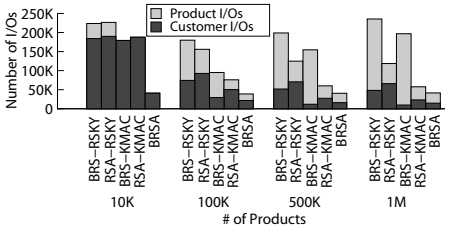


(b) CPU cost vs. Dimensionality (UN)

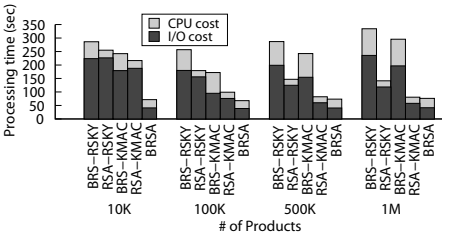
	2-D data		3-D data		4-D data		5-D data	
	BRS	RSA	BRS	RSA	BRS	RSA	BRS	RSA
I/O cost (sec)	20.7	23.3	179.9	156.0	687.6	555.0	1404	1161
CPU cost (sec)	18.4	5.5	76.9	23.2	1222.6	106.2	20042	428
Total cost (sec)	39.1	28.8	256.8	179.2	1910.2	661.2	21446	1589

	2-D data	3-D data	4-D data	5-D data
	BRS / RSA	BRS / RSA	BRS / RSA	BRS / RSA
I/O cost ratio	0.89	1.15	1.24	1.21
CPU cost ratio	3.35	3.32	11.51	46.81
Total cost ratio	1.36	1.43	2.89	13.50

(c) RSA & BRS costs vs. Dimensionality (UN)



(d) I/Os vs. $|P|$ (UN)

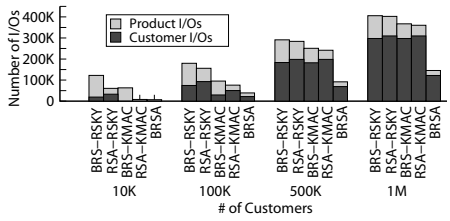


(e) CPU cost vs. $|P|$ (UN)

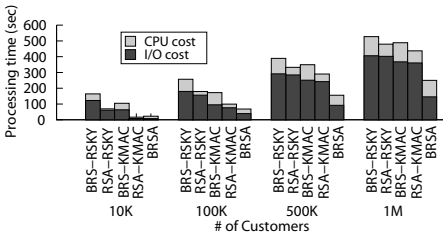
	10K		100K		500K		1M	
	BRS	RSA	BRS	RSA	BRS	RSA	BRS	RSA
I/O cost (sec)	223.6	226.5	179.9	156.0	198.9	125.0	235.5	118.5
CPU cost (sec)	62.7	28.6	76.9	23.2	87.9	22.0	98.8	22.9
Total cost (sec)	286.3	255.1	256.8	179.2	286.8	147.0	334.3	141.4

	10K	100K	500K	1M
	BRS / RSA	BRS / RSA	BRS / RSA	BRS / RSA
I/O cost ratio	0.99	1.15	1.59	1.99
CPU cost ratio	2.19	3.32	3.99	4.32
Total cost ratio	1.12	1.43	1.95	2.36

(f) RSA & BRS costs vs. $|P|$ (UN)



(g) I/Os vs. $|C|$ (UN)

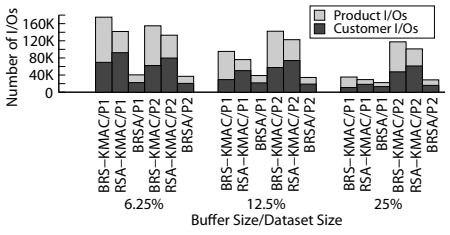


(h) CPU cost vs. $|C|$ (UN)

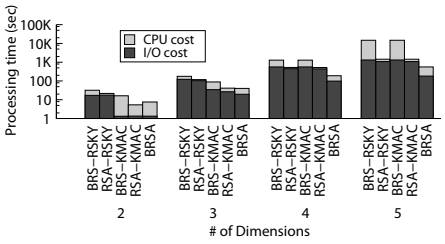
	10K		100K		500K		1M	
	BRS	RSA	BRS	RSA	BRS	RSA	BRS	RSA
I/O cost (sec)	122.4	60.6	179.9	156.0	291.0	284.1	405.8	402.4
CPU cost (sec)	41.6	8.5	76.9	23.2	98.1	48.1	121.5	77.0
Total cost (sec)	164.0	69.1	256.8	179.2	389.1	332.2	527.3	479.4

	10K	100K	500K	1M
	BRS / RSA	BRS / RSA	BRS / RSA	BRS / RSA
I/O cost ratio	2.02	1.15	1.02	1.01
CPU cost ratio	4.91	3.32	2.04	1.58
Total cost ratio	2.37	1.43	1.17	1.10

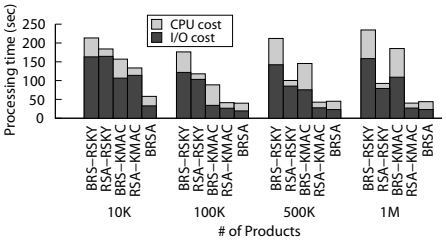
(i) RSA & BRS costs vs. $|C|$ (UN)



(j) I/Os vs. Cache Size (UN)



(k) CPU cost vs. D (AC)



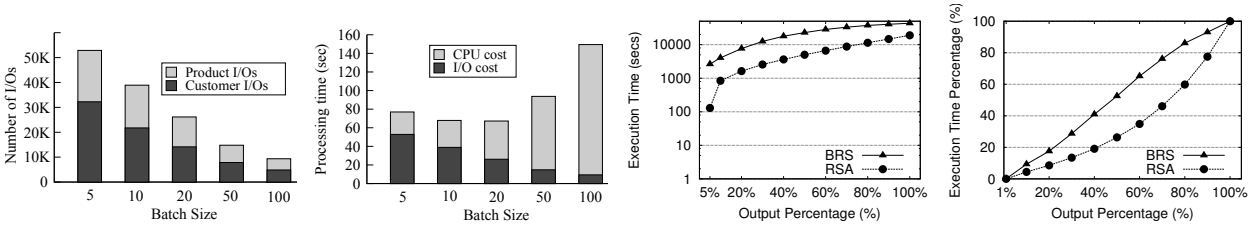
(l) CPU cost vs. $|P|$ (AC)

Figure 6: Experiments with Synthetic Data

by RSA proves to be more efficient in terms of the total I/O accesses required. Moreover, w.r.t. the processing cost (Figure 6(e)), RSA is significantly faster than BRS, and scales better as $|P|$ grows larger. Again, the corresponding numbers for the BRS and RSA algorithms are also presented, for clarity, in Figure 6(f). Finally, our bRSA algorithm is the most efficient algorithm for the case of k-MAC queries remaining essentially unaffected by the size of the product data set. Figure 6(l) shows an analogous behaviour of the algorithms in anti-correlated data, with the times being slightly smaller, as shown by the scale of the y-axis. In Figures 6(g)-6(h) we then plot the I/O and CPU costs when varying the size of the customer data set. As illustrated, RSA and BRS require roughly the same number of I/Os for large numbers of customers. This is predictable since RSA processes one customer entry per iteration, i.e., the number of iterations required by RSA is $O(|C|)$. Therefore, in the case when $|C|$ is much larger than $|P|$, the visiting strategy followed by BRS would be a more reasonable choice. However, even in this worst case scenario, RSA exhibits better overall performance than BRS algorithm, due to the significant lower processing cost (Figures 6(h)-6(i)). Again, our bRSA algorithm is notably faster than both single point algorithms.

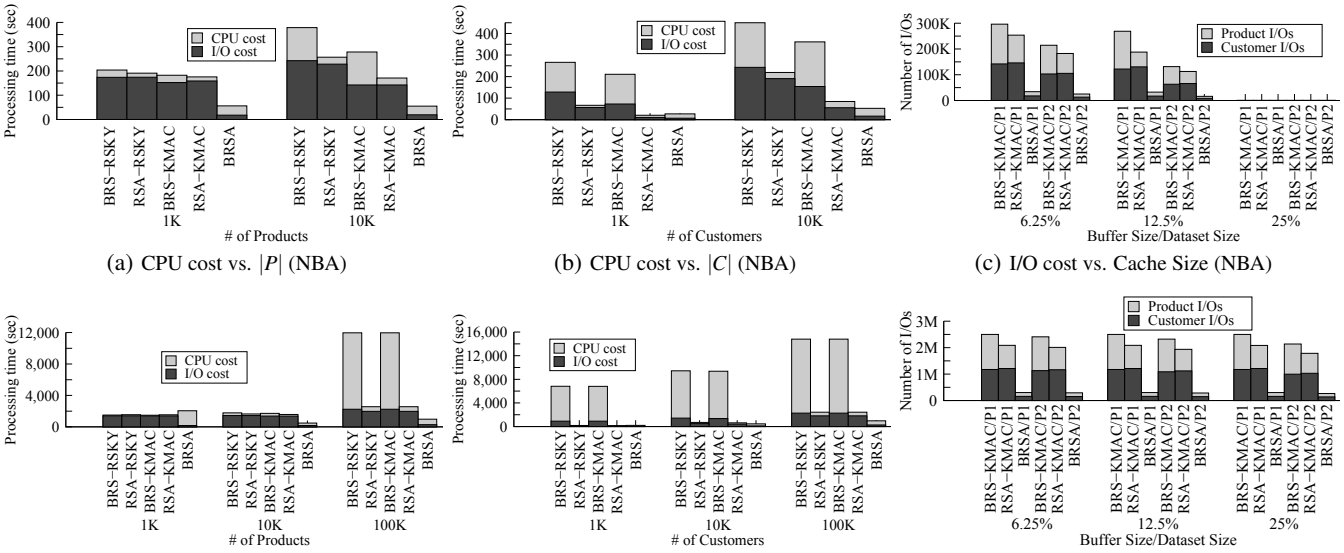
Sensitivity Analysis vs. Memory Size. For this experiment we compare the number of page accesses required by each algorithm with respect to the memory size allocated for caching. We varied cache (buffer) size from 50 pages (corresponding to 6.25% of the data set size) up to 200 pages (25% of available memory). We also experimented with two different cache replacement policies. For the first policy, namely $P1$, we followed a LRU strategy. Additionally, motivated by the intuition that entries with higher levels in the R-tree will be accessed more frequently, we also used a buffer that maintains pages in descending order of their tree level ($P2$). Figure 6(j) plots the number of I/Os required for different cache sizes and cache replacement policies. As depicted, regardless of the memory size and strategy used, both RSA and bRSA algorithms are more efficient in terms of disk accesses. Further, notice that LRU was slightly more efficient for the cache size that we used in our default scenario (12.5% of the data set size).

Sensitivity Analysis vs. Batch Size. We then investigate the performance of our bRSA algorithm with respect to the batch size G. We set each batch to contain from 5 to 100 candidates and plotted the results in Figures 7(a)-7(b). As expected, larger batch sizes result to fewer total I/O operations, since more pruning can be



(a) I/Os vs. Batch Size (UN) (b) CPU cost vs. Batch Size (UN) (c) CPU cost vs. number of reported results (UN) (d) CPU cost/Total cost (%) vs. number of reported results (UN)

Figure 7: Varying the Batch Size (left), Progressiveness of Reported Results (right)



(a) CPU cost vs. $|P|$ (NBA) (b) CPU cost vs. $|C|$ (NBA) (c) I/O cost vs. Cache Size (NBA)

(d) CPU cost vs. $|P|$ (HOUSE) (e) CPU cost vs. $|C|$ (HOUSE) (f) I/O cost vs. Cache Size (HOUSE)

Figure 8: Experiments with Real Data

shared among candidates, whereas the processing cost increases. Interestingly, when the batch size becomes larger than a threshold, the total processing cost (cost of disk accesses plus CPU time) increases, due to the growing cost of maintaining all local priority queues and the significantly more dominance checks required thereof. As showcased by the experiments, keeping fairly small batch sizes ($\sim$10 candidates) maximizes the efficiency of bRSA.

Progressiveness. Finally, we compare the progressiveness of RSA and BRS algorithms on a workload consisting of $|Q|$ reverse skyline queries. x-axis represents the percentage of reverse skyline results found so far compared to the total influence score. y-axis plots the time required to report the corresponding percentage of results, both in absolute time (Figure 7(c)) and as a percentage of the total time spent (Figure 7(d)). Both figures demonstrate that RSA is notably more progressive than BRS, especially for reporting the first query results. In particular, RSA outputs the first 5% of the reverse skylines in 1/10th of the time needed by BRS, which can be particularly important for applications that require a quick response or when the complete output is not useful.

Experiments with Real Data. Figures 8(a)-8(c) and 8(d)-8(f) report our experimental findings on the NBA and HOUSE data sets respectively. The results are in accordance with our experiments on synthetic data sets. Moreover, the performance gains achieved by RSA are higher in real data sets (especially vs. $|C|$), partly due to the higher data dimensionality (5 and 6 dimensions respectively), which results to more points belonging to the influence set (on average 196 points in HOUSE data set vs 11 points in UN data set).

7. RELATED WORK

Market research is a systematic, objective collection and analysis of data about a particular target market by taking into account factors such as products, competition and customers behavior. The work of [9] proposed formulating several economically incentivised applications (e.g., *potential customers identification*, *product feature promotion*, *product positioning*) as optimization problems taking a data mining perspective. In the context of database research, DADA [10] was the first of a series of works in the field proposing various queries by capitalizing on the dominance relationships among products and customer preferences.

Several works [5, 11, 25, 18, 6, 3] focus on identifying the potential customers of a product. In order to provide an insight on the product against the competition, [13, 24, 23] address the problem of discovering and promoting the best product features. Another practical application is how to design new products, such that they will maximize the expected utility, a problem known as product positioning [20, 19, 21, 12, 15]. The utility function may incorporate various factors such as the number of expected product buyers [19, 21, 12, 15], the actual profit (price minus production cost) [10, 20, 21, 15] or the number and features of other competitive products [10, 12]. Other works [20, 21] seek profitable packages formed by combining individual products, e.g., a flight and a hotel room, such that the profit gain of the package is maximized.

With regards to the number of expected customers, one problem is how to best model user preferences. One way is to assume that a weight vector capturing the importance of different product features (attributes) has been determined for each customer, through a

preference learning process. Based on this assumption, each product is assigned a score by applying the weight vector known for the user. Then, the products that score higher are those that would be more attractive to the respective user. This is the approach taken in *top-k queries* [7]. [18] tackled the reverse problem of discovering the most attractive products by introducing reverse top-k queries.

However, the weight vector formulation is often too difficult to come up with in real life [17]. A more natural way to model preferences is by allowing users to directly specify their preferred product attribute values. Taking this approach, both products and user preferences can be represented as points in a multidimensional space. In such a scenario, different notions of user satisfaction have been proposed. One option is to allow users to specify the worst acceptable value for each dimension [15]; all products having better values from the specified ones are considered as satisfactory. An important limitation of such a formulation is that it cannot be used for 'subjective' types of attributes. Further, there is no metric of how relevant each product is w.r.t. the actual user preferences. Thus, another option is to measure product attractiveness based on how close the product attribute values are to the user-preferred ones. In order to find the k most attractive products for a customer $c \in C$ we can issue a kNN(c) query [16] on the product data set. However, in several real applications it could be hard to find an appropriate distance function, because different dimensions might have different weights which depend on the preferences of each user.

With the goal to overcome the limitations of top-k and kNN queries, skylines have been widely used for multi-criteria decision analysis and for preference queries. The skyline query returns the set of not dominated objects, corresponding to the Pareto optimal set, which will always include the top-1 result for any monotone preference function. [4] introduces skyline queries in databases, also presenting various external memory algorithms. In order to capture subjective attributes, the dynamic skyline [14] returns all products that are 'attractive' according to a user's preferences.

Viewing the problem from a manufacturer's perspective, [5] introduces the reverse skyline query, which returns all customers that would find a product as 'attractive', and proposes a branch-and-bound extension of the BBS algorithm [14] that reduces the search space. [11] improves upon [5] by providing tighter pruning rules based on midpoint skylines (Section 2.2) and presents algorithms for calculating reverse skylines on uncertain data. [25] proposes the BRS algorithm (Section 2.3), which exploits additional optimizations for precise data. [6] considers non-metric attribute domains and proposes non-indexed algorithms to efficiently calculate the influence set in that case, whereas [22] studies how to process reverse skylines energy-efficiently in a wireless sensor network.

In this work, we formulate customers identification by following a reverse skyline approach, where we consider both subjective dimensions and competition. We focus on providing a more efficient and progressive algorithm for single-point reverse skylines. Further, we extend our methods for multiple query points, with applications to the k-MAC query. The methods proposed in [12, 15] cannot be applied onto our setting, because they assume the same product dominance relationships holding for all users and that they can be calculated by executing only one skyline query. However, this is not true when each user has his preferred attribute values.

8. CONCLUSIONS

In this work, we studied two classes of queries involving customer preferences with important applications in market research. We first proposed the RSA algorithm for reverse skyline query evaluation. We then developed a batched extension of our RSA algorithm that significantly improves upon processing multiple queries individually, by grouping contiguous candidates, exploiting I/O commonalities and enabling shared processing, and applied this batched extension to solve the k-MAC query. Our experimental study on both real and synthetic data sets demonstrates that (*i*) RSA is significantly more efficient, scalable and progressive than the BRS algorithm for the reverse skyline problem, and (*ii*) that our proposed batched algorithm is the best choice for the k-MAC query.

Acknowledgments This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund. The authors would also like to thank Prof. Dimitris Papadias for his support during the early stages of this research work.

9. REFERENCES

[1] http://randdataset.projects.postgresql.org.
[2] J. Bentley, K. Clarkson, and D. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA*, 1990.
[3] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Züfle. Inverse queries for multidimensional spaces. In *SSTD*, 2011.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
[5] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.
[6] P. Deshpande and D. P. Efficient reverse skyline retrieval with arbitrary non-metric similarity measures. In *EDBT*, 2011.
[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
[8] D. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *NRL*, 45, 1998.
[9] J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Journal of Data Mining and Knowledge Discovery*, 2(4), 1998.
[10] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada: a data cube for dominant relationship analysis. In *SIGMOD*, 2006.
[11] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.
[12] C.-Y. Lin, J.-L. Koh, and A. L. Chen. Determining k-most demanding products with maximum expected number of total customers. *TKDE*, 2012.
[13] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*, 2008.
[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1), 2005.
[15] Y. Peng, R. C.-W. Wong, and Q. Wan. Finding top-k preferable products. *TKDE*, 2012.
[16] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, 1995.
[17] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.
[18] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, 2010.
[19] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1), 2010.
[20] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *PVLDB*, 2(1), 2009.
[21] Q. Wan, R. C.-W. Wong, and Y. Peng. Finding top-k profitable products. In *ICDE*, 2011.
[22] G. Wang, J. Xin, L. Chen, and Y. Liu. Energy-efficient reverse skyline query processing over wireless sensor networks. *TKDE*, 24(7), 2011.
[23] T. Wu, Y. Sun, C. Li, and J. Han. Region-based online promotion analysis. In *EDBT*, 2010.
[24] T. Wu, D. Xin, Q. Mei, and J. Han. Promotion analysis in multi-dimensional space. *PVLDB*, 2(1), 2009.
[25] X. Wu, Y. Tao, R. C.-W. Wong, L. Ding, and J. X. Yu. Finding the influence set through skylines. In *EDBT*, 2009.