Journal of Theoretical and Applied Information Technology
20th October 2015. Vol.80. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFICIENT REVERSE SKYLINE ALGORITHM FOR DISCOVERING TOP K-DOMINANT PRODUCTS

[1]SHAILESH KHAPRE, [2]M.S. SALEEM BASHA, [2]A. MOHAMED ABBAS
[1]Department of Computer Science, Pondicherry University, Pondicherry, INDIA.
[2]Department of Computer Science and & Info. Technology, Mazoon University College, Muscat, OMAN
E-mail: [1]shaileshkhaprerkl@gmail.com, m.s.saleembasha@gmail.com, abbasasfaq@mazooncollege.edu.com

ABSTRACT

Recent boom in internet growth and the advancement in internet security have led to rapid growth in Ecommerce and related services. In this context, capturing the preferences of customers plays an important role in decisions about the design and launch of new products in the market. The science that primarily deals with the support of such decisions is the Operational Research. Since many of the research problems of Operational Research have to do with the analysis of large volumes of data, therefore there has been a keen interest in data management methods to solve these problems. In this work we develop new algorithms for two problems related to the analysis of large volumes of consumer preferences, with practical applications in market research. The first problem we consider is to find the potential buyers of a product (potential customer's identification). We formulated this problem as a reverse query skyline and propose a new algorithm called ERS. Secondly, Practical applications often require simultaneous processing of multiple queries. To resolve this problem, we formulated a new type of query, which is referred to as a query to find the k dominant candidates (k-dominant query). Our experimental evaluation validates the efficiency of the proposed algorithm which outperforms BRS by a huge margin.

Keywords: *Skyline Algorithms, Market Analysis, Personalized Service Mining, Personalized marketing, E-Commerce.*

1. INTRODUCTION

The development of methods for evaluating queries with preferences of particular interest of consumers is the on-going research in service provider's world. This is because in modern environment, the providers-companies are required to attract consumers with different characteristics and consumer habits, following them as closely as possible, learning and optimizing there personalized strategies. In this context, capturing the preferences of customers plays an important role in decisions about the design and launch of new products in the market. Examples of important applications related to the analysis of consumer preferences are personalized advertising, market segmentation, product positioning etc.

For example suppose the marketing department of an IT company, has conducted a market survey to gather consumer preferences regarding the desired features of a laptop. Using the data gathered, the company wants to assess what consumers are more likely to buy any of the computer models offered. In this way, the company could adjust its advertising strategy to these users (for example, sending personalized e-mails or their special offers). Another interesting application is to identify the features that should have a new product in order to maximize the most popular among consumers. In this case, the company would choose to adjust the standards set by the planning department.

The science that primarily deals with the support of such decisions is the Operational Research. The object of Operational Research usually described as follows: Wanted to optimize a number of parameters to maximize a utility function (utility function). Since many of the research problems of Operational Research have to do with the analysis of large volumes of data, therefore there has been a keen interest in data management methods to solve these problems.

In this work we develop new algorithms for two problems related to the analysis of large volumes of consumer preferences, with practical applications in market research. Then we present briefly the specific problems which we focused. The first problem we consider is to find the potential buyers of a product (potential customer's identification). We formulated this problem as an reverse query skyline and we propose a new algorithm called ERS which predominates in performance compared to the most efficient algorithm has been proposed in the literature so far BRS [1], in relation to both the computational cost and with the disk access cost. Unlike the algorithm BRS, the ERS algorithm we propose is based on a different ranges of data processing, which allows significant improvements in relation to the speed of implementation, scalability, and progressive production results.

Practical applications often require simultaneous processing of multiple queries, for example, if a mobile phone carrier which maintains a database of subscribers, keeps information about the current program subscriptions for each client, and usage statistics such as the average monthly call duration, no. of text messages sent, data volume consumed etc. Moreover, suppose that the sales department has proposed the launch of a new series of television programs, the company would like to provide these programs that collectively will be more likely to attract the maximum number of subscribers. To resolve this problem, we formulated a new type of query, which will be referred to as a query to find the k dominant candidates (k-dominant query). Given a set of existing products on the market P, a set of consumer preferences C and a new set of candidate products Q, a k-dominant query returns a set of k candidate from the initial set of Q, so that the selected new products have collectively the estimated maximum total number of buyers. According to the formulation of the problem we propose a consumer is considered a potential buyer of a product if and only if the product is the result of a reverse skyline set or influence set, considering for each consumer as the best attribute values that are related to their preferences.

Two recent works [2, 3] consider a similar problem but only features the data that have a universally accepted total provision, such as price or weight of a laptop (in both cases, lower prices are always preferable) . If subjected to this requirement, the best records for a user are easy to obtain by performing a conventional skyline query. Therefore, the above studies are concentrating on greedy algorithms that seek the most profitable solution, combining the sets of potential buyers with each new product. In this paper, we generalize the notion of preferences of a consumer to cover also features for which there is an optimal objective value (i.e. it does not follow a total order value) such as screen size, processor type, operating system, etc. For example, a consumer may be interested in buying the portable laptop while someone else buying a laptop to replace the fixed personal computer. In such a case, it is expected the first consumer to prefer a laptop with a smaller screen (making some sacrifices in usability and applications that can it perform), while the second one with a larger screen (perhaps sacrificing ease of transportation).

For such traits, the application of the techniques proposed in the work [2, 3] requires the previously exported relations of domination that exist between the records for each customer as they are user dependent preferences. Therefore, this method would require the implementation of a dynamic query skyline [4] for each consumer, which is prohibitively expensive compared to the execution time that would be required. At the same time, if it chooses to execute one of the algorithms that have been proposed till now (simple) reverse skyline queries to evaluate a k-dominant queries, we would need to calculate the outcome of a reverse skyline query once for each new product candidate, which is also very costly in terms of execution time, especially for larger datasets. With this in mind, we adapt the algorithm ERS proposed for simple reverse skyline queries. The algorithm we propose for this problem significantly reduces the required runtime compared by processing each query separately by grouping similar questions appropriately, performing common accesses to disk, and allowing simultaneous processing of many queries. Finally, having drawn the entire reverse skyline for each query, we propose a greedy algorithm that computes the final solution for a k-dominant product.

Overall contributions of the paper are:

- We propose a new algorithm, called ERS for evaluating reverse skyline queries. ERS algorithm shows better scaling to datasets containing a large number of results belonging to the ridge (e.g. in case of multidimensional data) while producing the first results significantly faster than the best algorithm has been proposed in the literature to date(BRS).
- We develop a variant of the ERS algorithm for processing groups of queries which significantly reduces the required runtime compared by processing each query individually appropriate grouping similar products candidates, performing common accesses to disk, and

allowing simultaneous processing of multiple queries. We then apply this new algorithm for evaluating k-dominant queries, the k-Dominant queries generalizes similar queries that have been proposed in previous work [2, 3] for cases where consumer preferences also include subjective attributes.

- Experimenting on synthetically generated data, outcome shows us that (a) the ERS algorithm significantly outperforms the BRS algorithm for the case of an reverse skyline query in relation to the performance, the scalability, and progressiveness, particularly for multidimensional data or when the size of the whole product dataset is larger than the size of all consumer preferences, and (b) the algorithm we propose simultaneously perform multiple queries which outperforms basic methods that process each item separately.

2. RELATED WORKS

The work [5] first proposed the formulation of several problems in operational research, such as finding potential market, potential customer identification, product-feature promotion and product positioning.

In the field of database the work [6] suggests different types of queries for market analysis, analysis of dominance relationships among competing products and consumer preferences. Goal is to help a company to place its products in perfect position as per the consumer choices in the market so to attract as many customers as possible.

Several works [7, 8, 1, 9, 10, and 11] focus on the problem of finding potential buyers of a product. To highlight the characteristics of a product relative to the competition, the work [12] focuses on the problem of defining and promoting the attributes of a product to make it competitive in comparison with other products. In [13] the problem of promoting a product is transformed into a problem of finding attractive in buyer i.e. dimensions subsets for which the product has good ranking. A corresponding problem is to find the top-k best areas to promote a product [14].

Another practical application relates to the design of new products in order to maximize the estimated benefit (utility function), a problem commonly known as the optimal placement of a product [15, 16, 17, 2, 3], the utility function may include various number of factors including the estimated buy [16, 17, 2, 3], the final gain (product price minus production costs) [6, 15, 17, 3] or competitions [6, 2]. The work [15, 17] deals with the problem of finding competitive packages composing individual product offerings, for example prices of flights with hotel rates. A package is competitive if not dominated by other packages.

Focusing on utility functions related to the number of expected buyers, a challenge is how to represent the preferences of consumers. A popular method assumes the existence of a weighted aggregate function that reflects the relative importance of each feature for a particular consumer. Based on this assumption, each product is assigned a score by applying the preference corresponding aggregate function on product prices. Products receiving the highest scores are considered as the most attractive for the individual user. The work [9, 16] follows this approach by introducing the notion of reverse top-k queries. A reverse top-k query returns the aggregation function for which (vectors) a product q belongs to top-k.

Nevertheless, in practice it is often quite difficult to get the exact aggregation function for each user of the system [18]. In contrast, a more natural way to represent preferences is allowing users to directly determine the ideal for those features for a product. Following this approach, both product and consumer preferences can be represented as points in the same multidimensional space. Such cases lead to different ways of measuring the satisfaction of a consumer-product. One option is to allow users to specify a minimum acceptable value in each dimension-feature [3], Based on this approach, all products that have better features than the minimum acceptable can be regarded as satisfactory. A major limitation of such a formulation is that it can be applied to types of features that do not have full rank, but the optimum value is subjective per user. Moreover, it gives a sense of satisfaction to consumer-product.

Therefore, another option for measuring satisfaction is based on how close is product-features to the user's needs. Based on this logic, one can identify the top-k most attractive products to a consumer by means of a query nearest neighbor. Given a set of point P and a point g, a nearest neighbor query (Nearest Neighbor (NN) query) [20] returns the point $p \in P$ that has the smallest distance from q, But even in this case, many times it is difficult to determine an appropriate distance function, mainly because each dimension has different weight depending on the preferences of a user, or because each dimension usually has its own unit of measurement.

Specifically to address some of the above limitations, the skyline Query have been widely used for research applications market. Suppose user preferences and characteristics of the products belong to the sets C and P respectively. In this case the meaning of the reverse skyline query to a point q which is introduced in [7] is used to find all of the points p ∈ P for which q belongs to the dynamic skyline. In other words, such a query returns all customers c ∈ C for which a specific product q is attractive. In this paper we propose a variant of the algorithm BBS [4] used in conventional skyline query, with a view of pruning large part of the search space.

The work [8] improves the method of [7] by considering the problem where the preferences and characteristics of products is uncertain, proposing an algorithm that solves the original problem minimizing write-read operations from the disk. The algorithm BRS presented in this paper [1] suggests an optimizations method for definite data; the work [10] addresses the problem for the case of non-metric spaces and proposes efficient algorithms that compute the reverse skyline. In addition, the paper [19] studies the problem of energy efficient reverse skyline query in a wireless sensor network.

Finally, it can be argued that there is a link between the market analysis and work done on the problem of facility location planning. In Reverse Nearest Neighbor queries (RNN) [20], for a set of points Q, the query returns points q ∈ Q, for which a point s ∈ S is the nearest neighbor. Finding the k best s points to a predetermined region of space in order to maximize the number of points that have q as nearest neighbor is treated in [21], this problem is exactly analogous to the problem that is being addressed in our research.

3. PRELIMINARIES

In this we give some basic definitions for the reverse skyline queries. Under section 3.1and 3.2 we present the meaning of the whole and its hinterland and present some important properties that apply. Finally, section 3.3 describes the most efficient algorithm that has been proposed in the literature to date of reverse skyline queries (BRS).

3.1 Reverse Skyline Queries

Let P and C be two sets of records of a table in a database consisting a set of attributes $A = \{A_1, \ldots, A_D\}$, alternatively consider each record as a point in a multidimensional space with D dimensions. In small letters we will denote a record that belongs to the corresponding set of capital letters, e.g. $p \in P$. We will refer to each record $p \in P$ as a product, while p_i will denote the value of the attribute A_i for product p. In addition each record $c \in C$ represents the ideal characteristics of a product for a consumer; we will refer briefly to each record for a consumer as c. For example, suppose that the database records contain information about the features of available models of laptops. In this case, possible attributes (dimensions) may be the price, weight, screen size, memory size, etc. some of the features are objective attributes i.e. e.g. best prices. For example, between two models with identical features, a buyer will always prefer the cheaper or lighter model. But it is clear that for some traits the optimal value is given by subjective preferences. For example, a large screen is more practical but also hinders the portability of a computer. Similarly, a very fast processor typically causes more heat and noise while reducing self-sufficiency. Some buyers prefer a powerful and larger sized laptop (desktop replacement laptop), while others a model with less power and smaller size (e.g., netbook or tablet). Obviously, a buyer is more likely to be interested more for products that fit quite to his liking. Given the existence of subjective preferences for attributes of a relationship, then we present the definition of the dynamic dominance, as given in [7].

Definition 1: Dynamic Dominance.

Consider two points $c \in C$, p, $p' \in P$. We will say that a product p dominates dynamically on a product p' regarding preferences of a consumer c, and will be denoted by $p \leftarrow {}_c p'$, if for each valid dimension $|p_i - c_i| \leq |p'_i - c_i|$ and there is at least one dimension such that $|p_i - c_i| < |p'_i - c_i|$.

Note that this definition can cover the objective dimensions, making the assumption that smaller values are preferred and preference is set to the minimum value for the attribute A_i. For example, assuming that a lighter computer is always preferable we simply assume that all customers apply the preference $c_{\text{weight}} = 0$, then we present the definition of a potential dynamic skyline query (from [7]).

Definition 2: Dynamic skyline query.

A dynamic skyline query relative to the preferences of a consumer $c \in C$, which is denoted as $SKY(c)$ returns all the products $p \in P$ which are not dynamically dominated with respect to c by some other product $p' \in P$.

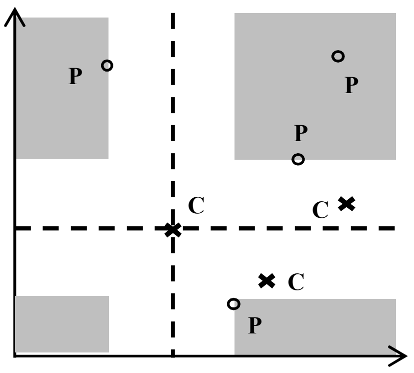

Figure 1(a). Dynamic Skyline Query relative to the c_1

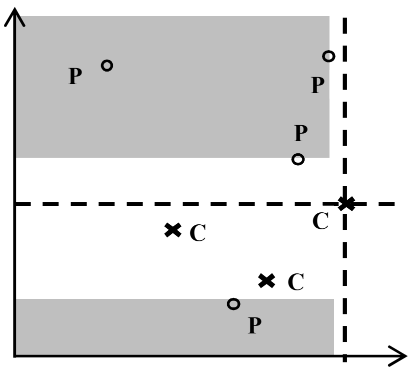

Figure 1(b). Dynamic Skyline Query relative to the c_2

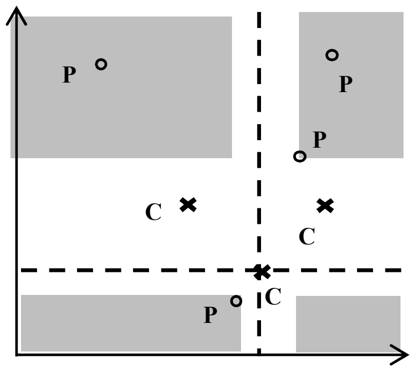

Figure 1(c). Dynamic Skyline Query relative to the c_3

Product **O** p_i
Customer **X** c_i
SKY(c_1) : {p_2, p_4}
SKY(c_2) : {p_1, p_3}
SKY(c_3) : {p_1, p_2}
RSKY(p_1) : {c_2, c_3}
RSKY(p_2) : {c_1, c_3}
RSKY(p_3) : {c_2}
RSKY(p_4) : {c_1}

Figure 1(d). Skyline Queries and Amount of Influence
Figure 1. Example: Dynamic Skyline Query

Suppose there is a set of products $P = \{p_1, p_2, p_3, p_4\}$, and a set of consumers $C = \{c_1, c_2, c_3\}$. Figure 1 (a) shows a Dynamic skyline query relative to the point c_1 considering two dimensions: processing power and screen size. The query result consists of products p_2 and p_4. Sections of the figure in gray areas are dominated by dynamic points belonging to dynamic skyline with respect to c_i. As we are interested only in absolute difference between the values of an attribute, a product may be dominating against a product that is in a different quadrant relative to the c_1. For example, the points p_1 and p_3 in the upper right quadrant dominates point p_2 belonging to the lower right quadrant, and the processing power and screen size are both closer to the preferences p_2 of c_1 from the corresponding characteristics of points p_1 and p_3. Figures 1 (b) and 1 (c) shows dynamic skyline queries on the points c_1 and c_3 respectively.

Then we present the problem from the reverse side, i.e. a product, quoting the definition of a Bichromatic reverse skyline query as given in [48].

Definition 3: Bichromatic reverse skyline query.
Let P and C be two sets of products and consumers respectively. A dichromatic reverse skyline query with respect to a point $p \in P$, which will be denoted as $RSKY(p)$ returns all points $c \in C$ such that $p \in SKY(c)$ the point p belongs to the dynamic skyline query relative to the point c.

In other words, a Bichromatic reverse skyline query with respect to a product p returns all consumers $c \in C$ who find the point p as attractive. Then we refer to the result of a Bichromatic reverse skyline query with respect to p as the total influence (influence set) with respect to p. In Figure 5.1 (d) shows the influence of sets of points p_1, p_2, p_5 *and* p_4.

The size of the total influence of a product p, $RSKY(p)$, can be seen as a way of measuring the impact of the product on the market. We refer to the size of the total impact of a product p, $|RSKY(p)|$, as scores of influence and is denoted as $IS(p)$. In the example of Figure 1, $IS(p_1) = IS(p_2) = IS(p_3) = IS(p_4) = 1$.

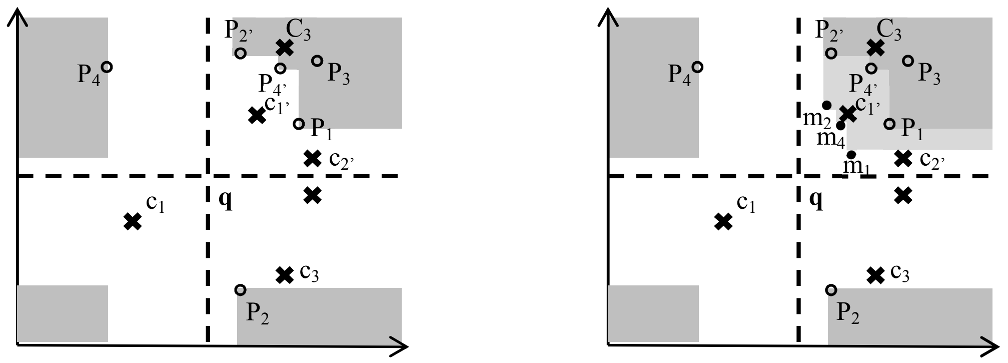

Figure 2(a). Transformed space 1st quadrant Ω_0 relative to the point q

Figure 2(b). Intermediate points (mid-points) relative to the q

Figure 2. Influence of neighbors as to q

3.2 Influence Region

Consider a point q (product) this point divides a space D to 2^D quadrants Ω_i, wherein each is determined by a number in the range $[0, 2^D - 1]$. In the example of Figure 2 where D = 2, the point q divides the space into four quadrants. As in the case of dynamic skyline queries we are interested only in the absolute difference values of attributes, we can transform points from all quadrants in the first quadrant Ω_0 as shown in Figure 2 (a), for ease of presentation, and then we focus on the first quadrant Ω_0 points if it relates q.

For each point p_i belonging to the skyline of q, and $m_i(q)$ the mid-point connecting q with p_i. In Figure 2 (b), the points are represented with black spot m_1; m_2 and m_4 representing the midpoints of p_1, p_2 and p_4 relative to q. so, whenever we will mention a product p_i that would mean the corresponding intermediate point $m_i(q)$ relative to q. Also we will assume that each point p_i can be calculated with a simple calculation on the corresponding $m_i(q)$ when processing a skyline query.

Uniting all the regions space that are not dynamically dominated in terms of q via skyline of q, showing the so-called influence region for q, which we denote by $IR(q)$. In Figure 2 (b) the non-shaded region in the first quadrant Ω_0 shows the influence *region* of q. Please note that items belonging to the skyline are not in same part of the influence region, as according to the definition of dominance, two points with equal values in each dimension is not dominated by one against the other. So we present a useful property that applies in relation to the influence region, as described in [8].

Property 1. A point (consumer) c belongs to the influence set $RSKY(q)$ of product q if and only if the point c belongs to the influence region q, such that $c \in IR(q) \Leftrightarrow c \in RSKY(q)$.

Returning to the example of Figure 2 (b), we observe that only the point c_2 belongs to the $IR(q)$. Therefore RSKY(q) = {c_2}.

Then we will assume that all the points (either owned by set of products or the consumer) is indexed by means of an R-tree. In an R-tree, the points with similar values of attributes (dimensions) are grouped and assigned to nodes. Each node contains a minimum bounding box (MBB) which encloses a number of points whose actual values are unknown to the particular node. Figure 3 (a) shows an MBB e. min-corner will be denoted by $e^-(q)$ the top of the MBB which is the minimum Euclidean distance from the point q. Among the points of MBB, point $e^-(q)$ dominates the larger piece of the multidimensional space. Furthermore, due to the way of construction of a MBB which is the smallest possible rectangular parallelepiped that includes a set of points, each side of the MBB must contain at least one point. At worst this point may be located in one of the peaks. We will refer to the peaks belonging to the sides of MBB as d and which is exactly diametrically opposite of q as minmax-corners. Each MBB contains exactly d such peaks. Regardless of the region of points within a MBB e, such that each point e dominates the area of its minmax-corners, while mostly dominating in the region dominated by the min-corner.

Given a set MBB we can extract two sets: a set of L containing all the min-corners and a set U that contains all the minmax-corners relative to q.

Figure 3 shows an example for all nodes $E_P = \{e_{p_1}, e_{p_2}, e_{p_3}, e_{p_4}\}$ where e_{p_i} denotes a node that contains points-products. In Figures 3 (b) and 3 (c) parallelepipeds represent midpoints and points with black and hollow dots show the respective min-corners and minmax-corners. Continuing the example of Figure 3 (b), the gray area corresponds to a lower limit of influence region of q, $IR^-(q)$, and is defined as the area that is not dominated by any of the min-corners belonging to set L. Similarly, the gray area in Figure 3 (c) corresponds to an upper limit of influence region of q, $IR^+(q)$, and is defined as the area that is not dominated by any minmax-corners belonging to set U . According to the work [1] applies the following property:

Property 2. If a node e_c dominated from a point $u \in U$, i.e. the node e_c is completely out of the upper limit of influence region $IR^+(q)$, then the node e_c cannot contain any point within the (effective) area of influence IR(q). Therefore, in accordance with the property 1 e_c node can be pruned.

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

3.3 The Brs Algorithm

Here we present briefly the most efficient algorithm proposed so far in the literature of reverse skyline queries, the Bichromatic Reverse Skyline - BRS [1]. BRS algorithm aims at minimizing the number of input/output (I/Os) by (a) progressively limiting influence region of q extracted until the final influence region, and (b) applying the property 2 to prune some nodes that do not contribute to the total influence RSKY(q).

The BRS algorithm uses two R-trees T_P and T_C which indexes data sets P and C respective. In addition for each dataset it maintains a priority queue (heap) E_P and E_C respectively, which are classified on the basis of the Euclidean distance of each node from the point q. The BRS algorithm runs in iterations. Initially, the algorithm adds the root of the tree T_P (respectively T_C) in the tail. In each iteration the BRS algorithm produces sets L and U consisting of all min-corners and all the minmax-corners each $e_p \in E_p$. Then we calculate the skyline of sets L and U, which we denote as SKY(L) and SKY(U) respectively.

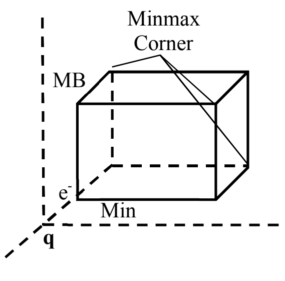

Figure 3(a). example MBB

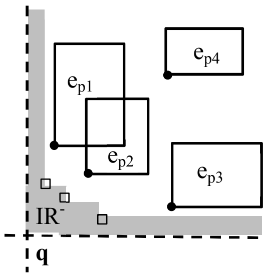

Figure 3(b). Lower limit of the influence Region IR (q)

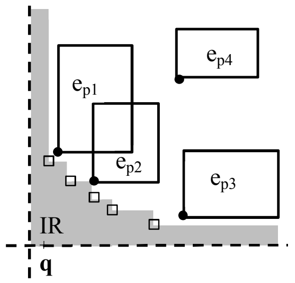

Figure 3(c). Upper limit of the influence Region IR (q)

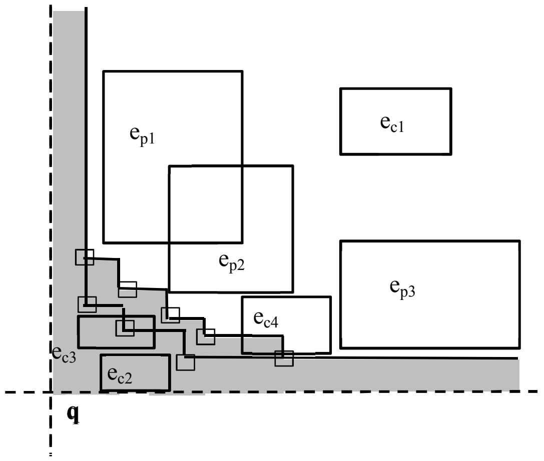

Figure 3(d). Example node pruning

Figure 3. Influence Region

In each iteration, the algorithm removes the priority queue E_P node with the smallest Euclidean distance from the q and update appropriate current sets L and U and the corresponding sets of skyline SKY(L) and SKY(U). Then, for each node $e_c \in E_C$ it verify dominance by SKY(L) and SKY(U). If the node e_c is not dominated by SKY(L), i.e. the section has a lower limit of the influence region of q, $IR^{-}(q)$, then the algorithm accesses the node e_c as it is likely to contain points that lie within the zone of influence $IR(q)$. In the example of Figure 3 (d), node e_{c3} intersects the $IR^{-}(q)$, therefore should be accessed and the child node is added to the priority queue. Conversely, if a node e_c is dominated by SKY(U) (such as e.g. node e_{c1} in Figure 3 (d)), then the node can be pruned e_c on the basis of property 2. BRS algorithm terminates when queue is empty E_C, i.e. when the exact position of each node is determined either within the influence region IR(q) so that all the leaves of the subtrees belongs to the reverse skyline relative to q, and if outside the influence region IR(q) then rejected from the result.

3.3.1 Limitations Of The Brs Algorithm

Complexity analysis: Let p_k and c_k current counterparts on priority queues E_P and E_C during the kth iteration of the algorithm BRS. In the worst case sizes of p_k and c_k are equal to the sizes of the datasets |P| and |C|. As described above, each iteration of the BRS algorithm maintains a totals skyline SKY(L) and SKY(U) having a size O(|P|) and O(D|P|), respectively, where D is the number of dimensions. The BRS algorithm performs dominance checks between each node belonging to the E_P and the E_C sets of skyline SKY(L) and SKY(U). Therefore, each iterations requires $O\left(D|P|X\ (|P| + |C|)\right)$ dominance checks, or otherwise $O\left(D^2|P|\ X\ (|P| + |C|)\right)$ comparisons, since each check takes $O(D)$ comparisons.

As its clear from the above analysis, BRS algorithm depends essentially on the size of sets of skyline SKY(L) and SKY(U). The work [22] shows that for uniformly distributed data, the size of the skyline is $\Theta(\frac{(\ln|P|)^{D-1}}{D!})$. Therefore, the processing cost of BRS algorithm is essentially unaffordable for larger data sets or higher dimensionality. Our experimental evaluation confirms the above analysis. More specifically, our experiments show that the processing cost of the BRS algorithm increases drastically for $|P| \geq 10^6$ or $D \geq 4$.

In order to tackle the problem of scalability in valuation of reverse query skyline, we suggest a more efficient algorithm called ERS which basically avoid calculating totals skyline SKY(L) and SKY(U) and therefore behave better for different dimensions and more generally for data with large size skyline.

Order Processing: The algorithm runs asynchronously following the R-trees T_P and T_C, a layout based on the Euclidean distance of each node from the point q. This processing order ensures minimized I/Os required on the index T_P. However, focusing on the total number of I/O operations required, the algorithm BRS performs sometimes some unnecessary I/Os. Figure 4 (a) illustrates such a case where the nodes e_{p2} and e_{cl} have not yet been accessed. In the next step the algorithm BRS access node e_{c1} and if not affected by this access therefore should be accessed later. Conversely, if the first node of accessing e_{cl}, the algorithm finds specific descendants nodes - e_{c2} and e_{c3} who dominated the points p_1 and p_5 respectively then the node is pruned. This avoids the additional access node e_{p2}. The proposed ERS algorithm follows a different processing order primarily based on the level of each node in T_C, as confirmed experimentally that the proposed ordering requires less overall input/output.

Progressive production effects: The BRS algorithm progressively refines the boundaries of influence region IR(q) and returns all points belonging to the set C, and within the lower limit $IR^{-}(q)$, Because of the processing sequence that BRS follows, it usually takes several iterations until to find the first results, which is not practical for applications that require only a part of the result set or require quick response. The ERS algorithm attempts to address this problem by producing first results fairly quick as of BRS.

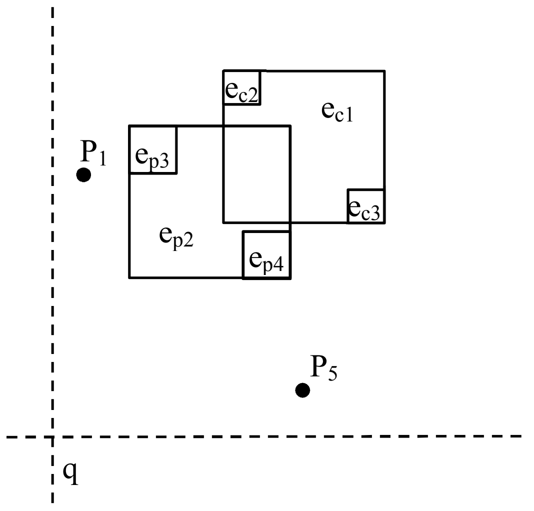

Figure 4(a). The BRS algorithm accessing node e_{p2} performing a redundant I/O

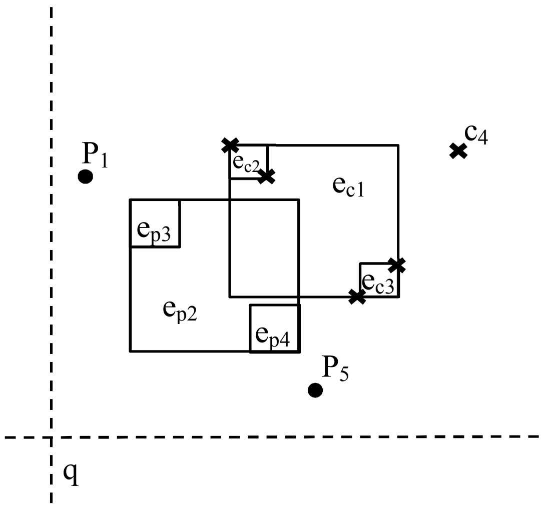

Figure 4(b). The ERS algorithms avoids accessing the Node e_{p2} lopping the c_4 using the point p_1

Figure 4. Order processing and disk accessing

4. THE ERS ALGORITHM

We present the algorithm ERS (Efficient Reverse Skyline Algorithm), which aims to address the problems described above.

General idea: ERS algorithm

- Avoids calculating SKY (L) and SKY (U) at each iteration, and therefore has a lower processing cost.
- Each iterations examines a node from the priority queue E_c following a series of processing based on two criteria: (a) the level of the node in the tree, and (b) the Euclidean distance of the node from the point q.
- Accessing a node from the priority queue E_P only if required to determine whether a point of C have total influence RSKY(q).

The ERS algorithm maintains two priority queues EP and EC and a set SKY(q) containing midpoint skylines that have been found to the current iteration. The two priority queues are classified according to two criteria: firstly based on the level of the node in the corresponding R-tree, and secondarily based on Euclidean distance of the node from the point q. Thus, the leaf nodes having higher priority are dealt first, while the operations on intermediate nodes are prolonged for later. Following this order of processing a leaf node may reveal some midpoint skyline which then can be used to prune an intermediary node e_c on basis of the Property 2. The same logic applies for nodes e_p as a midpoint skyline can also be used to prune an intermediary node e_p if the node e_p does not contribute to the skyline. Initially, the current node e_c is examined for dominance by SKY(q) and then with all the leaf nodes that belong to the priority queue E_P. If there is no leaf that dominates the node e_c, then the ERS algorithm accesses the next intermediate node e_p. This change in the order processing reduces the number of I/Os. For example, Figure 4 (b) the ERS algorithm considers the first leaf node p_1 and discovered that the particular node dominates the c_4, so that point c_4 does not belong to the influence set $RSKY\ (q)$. This avoids the access node e_{p2}.

4.1 ALGORITHM DESCRIPTION

Initially the ERS algorithm adds descendant nodes to the queue T_P (respectively T_C). Then the algorithm runs in iterations. In each iterations the ERS removes a node from the queue T_P (line 5) and checks the following pruning conditions:

- If node e_c is dominated by some point that belongs to the current total of midpoint skyline, SKY(q), then node e_c is rejected by the results based on the property 1 (lines 6-8).
- Else if the node is an intermediate node e_c (line 9), then the node is accessed and e_c-descendants nodes is added to the queue E_C (line 10).
- Otherwise, for each node e_p belonging to the queue E_P (lines 12-22):
- If node e_c is dominated by a midpoint such that $e_p \in E_P$ (line 15), then the node e_c is rejected based on property 1, while midpoint for e_p is added to the total Sky(q) (line 17).
- Otherwise if the node e_c is dominated by the midpoint of min-corner e_p^- such that $e_p \in E_P$ (line 20), then the node e_p and the nodes descendants are added to the queue E_P (line 21).

Finally, if node e_c is not pruned by any of the above conditions (line 23), then e_c belongs to the reverse Skyline of q and can be returned directly as a result (line 24). The ERS algorithm terminates when the priority queue E_C is empty so the total influence RSKY(q) is returned (line 25).

```
*Algorithm 4*: ERS
**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. $q$
**begin**
$SKY(q) := 0$; $RSKY(q) := 0$;
**while** $E_C = 0$ **do**
dominated := false;
$E_C(q).pop() \to e_c$ ;
$if\ dominated\ (e_c, SKY(q))$ **then**
$dominated := true$;
$continue$;
**if** $e_c$ is a non − leaf entry **then**
$Expand\ e_c, insert\ children\ entries\ in\ E_C(q)$;
**else**
*foreach* $e_p$ **£** *Ep (q) do*
 midpoint(ep, q) → m;
*if* $e_c$ is dominated by *m then*
*if* $e_p$ is a leaf entry *then*
*if* (dominated(m, SKY(q)) == false) *then*
SKY (q). push(m);
$dominated := true$;
break;
else
Expand $e_p$, insert children entries in $E_P(q)$;
$E_P(q)$. remove($e_p$);
**if** (dominated =
= false) **then**
RSKY(q). push($e_c$);
return RSKY(q);
```

Example 1. The performance of ERS algorithm will express better with the help of the example of Figure 5.5. First we have $E_P(q) = \{e_{p7}, e_{p1}, e_{p4}\}$ and $E_C(q) = \{e_{c1}, e_{c4}\}$ (nodes at the same level classified so as to the Euclidean distance from the point q). During the first iteration algorithm ERS will examine e_{c1} node which has the minimum distance from q. Because they are intermediate node ERS accesses node e_{c1} (line 10) and adds - descendants c_2 and e_{c3} nodes to the priority queue $E_C(q)$ (see Figure 5 (b)). At this point we have $E_C(q) = \{c_2, e_{c4}, e_{c3}\}$ so the algorithm chooses node c_2. Even after processing total skyline, if node c_2 is not dominated by any point-product the algorithm continues to check whether the node c_2 is dominated by a node contained in the queue $E_P(q)$. As node c_2 is dominated by the min-corner of the first node in the queue $E_P(q)$, e_{p7} (line 14). Therefore the algorithm examine whether there is a point (leaf node) in e_{p7} which prevails against the c2 as to q. For this reason, the algorithm accesses the node e_{P7} (line 21), adding - descendants p_8 and p_9 nodes queued to $E_P(q)$. Plus we have $E_P(q) = \{p_8, p_9, e_{Pl}, e_{P4}\}$ (see Figure 5 (c)). At this point the algorithm finds that node c_2 is dominated by the point p_8. Therefore, the point p_8 is added to the skyline (line 17) and node c_2 is discarded from the results. In the next iteration, the ERS algorithm will consider node e_{c4}. The node e_{c4} is not dominated by the current set after the skyline is accessed, the intermediate node and the descendants node c_5 and c_6 are added to the queue $E_C(q)$ (line 10) (see Figure 5 (d)). Plus we have $E_C(q) = \{c_5, c_6, e_{c3}\}$. Then the algorithm will consider node c_5. Again node c5 is not dominated therefore returned to the result (line 24). Then the ERS algorithm examines node c_6 which is dominated by point p_8 belonging to the current skyline (line 6) and is therefore rejected. In the last iteration the algorithm will consider node e_{c3} and node e_{c3} is dominated by point p_8 and therefore rejected. The priority queue $E_C(q)$ is now empty so the ERS algorithm terminates and returns the point c_5 as the final result of the query.

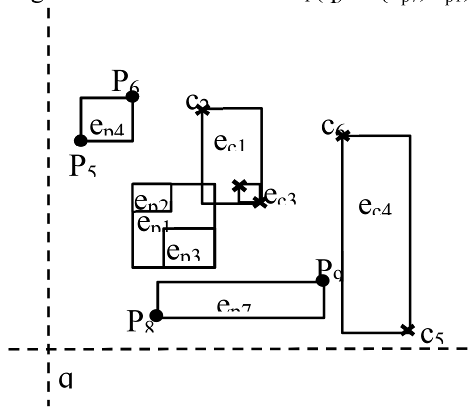

Figure 5(a). step 1

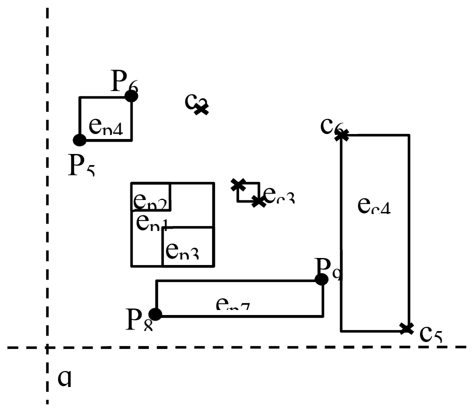

Figure 5(b). step 2

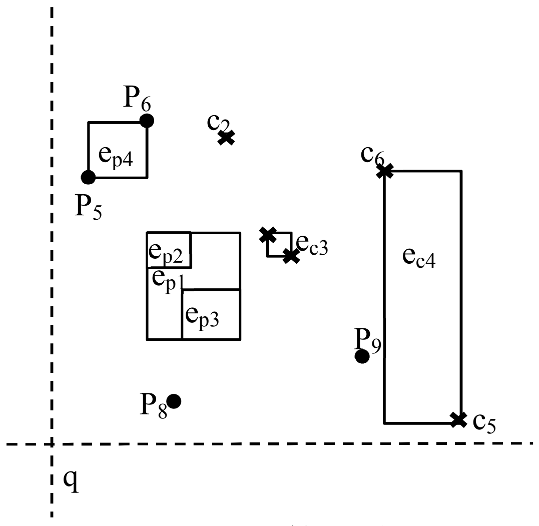

Figure 5(c). step 3

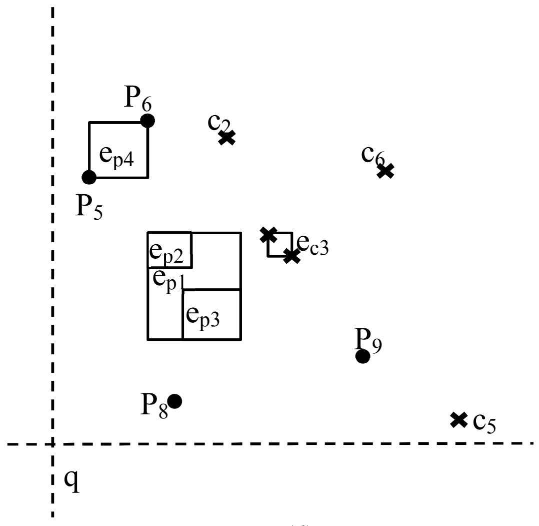

Figure 5(d). step 4

Figure 5. Example execution of the ERS algorithm

Complexity analysis: The ERS algorithm requires in the worst case is |C| iterations, one for each point in set C. Of course, certainly in practice many nodes will be rejected with the help of this set skyline SKY (q) (line 6). Each iteration comprises a check rule (a) with the current set of skyline SKY(q), and (b) with the set of nodes belonging to the priority queue E_P. Both sets have size O(|P|) at worst case so the total ERS algorithm requires O(|P| |C|) dominance check, or otherwise O(D|P| |C|) comparisons.

Progressive Production of results: Returning to the discussion on the progressive production results, we remind that the ERS algorithm always considers first leaf node that simultaneously have the minimum Euclidean distance from the point q. In other words this means that the first iterations cover points which are very close to q. Intuitively, the closer the point is in the set C w.r.t q, the more likely it is not dominated by any other point of the set P with respect to q. Therefore, the points considered in the first iterations have a high probability of belonging to RSKY(q). Furthermore, the criterion for classification in queue E_C, i.e. the first nodes to be examined will be leaf nodes and therefore will not need to be accessed by the respective nodes on disk, which means that the first iteration will generally be faster than the next. As described above the ERS algorithm takes comparatively little time to find the first results of a reverse skyline query. In contrast, we remind that the BRS algorithm requires several iterations to determine the influence region with such a degree of accuracy.

5. FINDING K DOMINANT PRODUCTS

We then propose a new type of query which will refer to as query to find the k dominant products. The k-most dominant query generalization problems have been studied in recent work [3, 2] covering the case where consumer preferences include subjective features. Initially we describe an example application that demonstrates the usefulness of such query. Then we will present the formal definition of query k-Dominant.

Example application of k-Dominant queries

Suppose a company manufacturer of portable computer which is planning to produce a new model series. To decide which models to choose to put into production, the company must take into account: (a) all competing models P that exist in the market, (b) total consumer preferences C that have been expressed as to the specifications of a model, and (c) all candidates new models Q as proposed by the design department. The objective is to determine the k models from all Q which is expected to have the greatest impact on the market, i.e. they jointly expected to attract the maximum number of buyers. To clarify that we call sub-k most attractive models, as it makes no sense to choose the result of two models which are expected to attract the same set of buyers.

5.1 Problem Definition

First we define the joint influence of all candidates for a set Q. Then we present the definition of joint influence scores and introduce the notion of query k-Dominant.

Definition 4. (Total Influence): Given a set P of products, a set of consumer preferences C and a set of new (candidate) Q, the set of joint influence of Q, which we denote as RSKY(Q), is defined as the union of the individual sets influence all $q_i \in Q$:

$$RSKY(Q) = \bigcup_{q_i \in Q} RSKY(q_i)$$

Based on the above definition, influence score IS(Q) of a set product candidates Q is equal to the size of the influence of the total Q, |RSKY(Q)|. Definition of a k-Dominant query as follows:

Definition 5. (Find the k-Dominant): Given a set P of products, a set of consumer preferences C, a set of new (candidate) Product Q and a positive integer k> 1, a query k-Dominant returns the subset $Q' \subseteq Q$; with size |Q'| = k which maximizes the influence score IS(Q').

Figure 6 shows an example query k-Dominant where we consider two existing (competitive) products p_1 and p_2 and 3 new candidate models. As shown, the result of a 1-Dominant query will return the model q_3 for which IS (q_3) = 2. Similarly, a 2-Dominant query will return models {q_2, q_3} which have jointly influence score 3.

It should be noted that it is possible that more than one candidate products are attractive based on the preferences of a consumer. For example, in Figure 6 (b) , both the q_1 and q_3 belong to SKY (c_2), furthermore it clarifies from the valuation of k-Dominant, that each product candidate $q \in Q$ is considered independently of the other nominated one only in comparison with the existing products of all P. This assumption is consistent with an actual application where a company is interested to compare all products only in relation to the competition. Also, at the end of this section we describe how we handle cases where two product candidates having equivalent influence scores.

Previous work [3, 2] attempt to address a similar question that has been proposed by inserting queries. However, this work only considers the case where all the features of database is objective, i.e. have a general optimum value (for example in the case of a laptop: market value, infinite range, etc.). The k-Dominant queries extend these contracts covering also subjective traits, where the optimal value depends on the preferences of each consumer. Moreover, in the work [57 , 49] it is assumed that the relationship between dominance and competing products candidates are known in advance, which is unfortunately only possible in case of objective attributes . Therefore, the contribution of the work [3, 2] merely propose efficient algorithms for selecting candidates that belong to the query result . In contrast, in our case the focus is on efficient identification of all influence.

5.2 K-Dominant Queries Variants

Based on the definition of a k-Dominant query we consider a product group only once. This is because two products with equal attribute values are in identical zones of influence. In a similar way we can handle a total consumers S_C with identical preferences, where $|S_C| > 1$, It suffices: (a) to consider only one consumer for each such group with weight equal to $|S_C|$ and (b) take into account the specific weight when calculating the joint influence score.

Another variation of k-Dominant query would be to correlate any potential buyer c_i belonging to total influence RSKY (q) with a weight w_i. The specific value for the weight w_i represents the probability that the consumer c_i eventually buy the product q. For example, a parameter that can be used to calculate the probability is the distance between the points q and c_i of the multidimensional space.

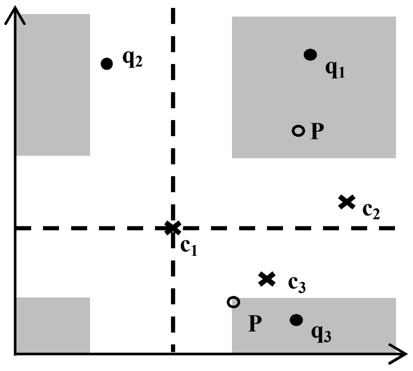

Figure 6(a). Dynamic Skyline of c_1

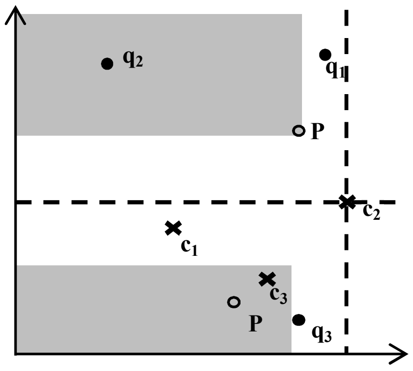

Figure 6(b). Dynamic Skyline of c_2

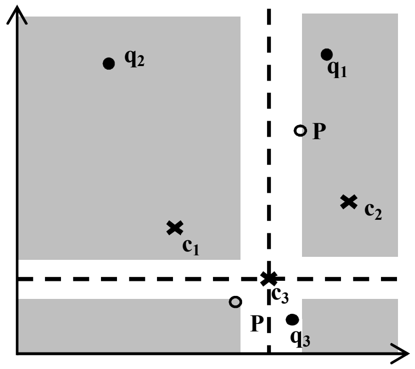

Figure 6(c). Dynamic Skyline of c_3

Product **O** p_i
Customer **X** c_i
Candidates q_i
RSKY(q_1) : {c_2}
RSKY(q_2) : {c_1}
RSKY(q_3) : {c_2, c_3}
RSKY(q_1, q_2) : {c_1, c_2}
RSKY(q_1, q_3) : {c_2, c_3}
RSKY(q_2, q_3) : { c_1, c_2, c_3}
2-Dominant{q_1, q_2, q_3}: { q_2, q_3}

Figure 6(d). Total influence sets

Figure 6. Example k-Dominant Query

5.2.1 K-Stage Selection Algorithm

Treatment of a k-Dominant is anything but simple. Specifically, the problem can be separated into two subproblems: (a) the calculation of the sub-assemblies that influence a set of candidate products, and (b) finding a subset of size k that maximizes profit measured as the sum of potential buyers (influence scores). In the next section we propose techniques for efficient processing of the first part. Considering the individual sets of influence known, the second subproblem can be transformed into a more general problem known as maximum k-coverage. This problem is NP-hard and therefore an exhaustive examination of all possible subsets of size k is not a feasible option. So, we propose an efficient greedy algorithm to solve the problem, the solution we propose is a variant of the more general k-coverage algorithm described in [31], As we ensure the following property, the profit generated by the solution is at most 1 - 1/e of the profit of the optimal solution.

```
Algorithm 2: k-stage Selection Algorithm
Input: Q a set of candidates, RSKY(q_i) reverse
skylines of qi, k
Output: Q′ the most attractive set of candidates
where |Q′| = k
begin
Q′ := 0; TempRSKY := 0; MaxRSKY := 0;
while |Q′| < k do
TempRSKY := MaxRSKY;
foreach q_i ∈ Q do
 if |RSKY(q_i) ∪ MaxRSKY| > |TempRSKY| then
TempRSKY := RSKY(q) ∪ MaxRSKY;
BestCand := {qi};
MaxRSKY := TempRSKY;
Q := Q — BestCand;
Q′ := Q′U BestCand;
return Q′;
```

Then we describe how we adapt the algorithm k-stage coverage with k-Dominant query, our algorithm (k-stage Selection Algorithm - KSA) takes as input a set of candidate products Q and returns a subset $Q' \in Q$, where |Q'| = k, which is a (1 - 1/e) - approximate solution of the k-Dominant query. KSA algorithm runs in iterations. In each iteration the algorithm examines all candidates product and chooses which if added to the current result would lead to the maximum possible increase in the joint influence score? If more than one candidate products have resulted in an equal increase in IS(Q'), then the algorithm KSA selects the product that has the minimum sum of distances from points belonging to all influence. The reason we choose this criterion is because the closer the specifications of a product to a consumer's preferences, the more likely the consumer will be interested in the purchase of this product. The KSA algorithm terminates after k iterations and returns the result set of Q'.

5.3 Processing Multiple Reverse Skyline Queries

k-Dominant Queries is an example query that demonstrates the need for simultaneous measurement of multiple inverse queries skyline. In this section we extend the ERS algorithm proposed for simple skyline query vice versa in the case of multiple queries.

The simplest method of processing multiple reverse skyline queries is to implement an algorithm for simple queries, such as BRS or ERS for each point. But this approach is very inefficient relative to the number of inputs/outputs of the disc needed. Specifically, several nodes $e_p(e_c)$ will need to be accessed many times since they appear in the priority queue more than once.

5.3.1 Algorithm Gers

Here we describe the algorithm gERS, which is an extension of the ERS algorithm proposed for the case of simple skyline queries, the main objective gERS algorithm is to reduce the total number of required I/O operations, by exploiting the possible proximity between candidates and allowing sharing a portion of the processing. It should be noted that the algorithm gERS we propose can be applied outside of k-Dominant queries and other types of queries that require processing of multiple reverse skyline queries.

The algorithm gERS process multiple queries in parallel, grouping them in such a way that the points are in a group to take advantage from the processing of other group members. The algorithm attempts to avoid unnecessary I/O operations using nodes accessed during the execution the ERS algorithm on a portion of query to prune nodes that belong to the priority queues of the other group members.

Specifically, when node is accessed the entering children nodes are updated in respective priority queue and simultaneously updating all priority queues of all members of the group containing the original node. Therefore, each node runs only one access per group. Moreover, in order to further improve the processing cost, the algorithm maintains a set gERS products (leaf corresponding R-tree) which are considered to prune large amount of space. Then we use the term vantage points to refer to these points, their use will be explained below and describe in detail with the implementation of the gERS algorithm.

At this point it is important to mention that the data structures needed to implement the algorithm gERS (e.g. priority queues, Skyline sets etc.) occupy a significant part of main memory. In the general case, particularly for larger |Q|, we can safely assume that all these data structures can fit in main memory. Based on the capabilities of our system, we will consider only G queries can be processed in parallel, where $G \ll |Q|$, as we will see in the experimental evaluation of the algorithm, the algorithm gERS displays optimal behavior by keeping the value G in relatively small size (e.g. up to 10 queries per group), the reason is that larger group sizes lead to explosive growth in the processing costs associated with the management of priority queues and their required dominance checks, which quickly offset the benefit from the reduced number of I/O operations.

```
Algorithm 3: gERA
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 $|G_j| = G$
begin
partition Q into $[|Q|/G]$ batches $\rightarrow G_j$;
foreach $G_j$ do
while
(RSKY($q_i$) for all $q_i \in G_j$ have not been found) do
selectCandidate $\rightarrow q_i$;
/* Process qi until IS(qi) has been completely determined */
if $E_C(q_i) \neq \phi$ then
BatchRSA($q_i$, $G_i$, $T_n$, $T_C$, $Ep\ (qi)$, $E_n(q_i)$, RSKY($q_i$), SKY
```

```
Function Group-ERA
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 $q_i$
begin
while $E_C(q_i) = 0$ do
dominated := false;
$E_C(q_i).pop() \rightarrow e_c$;
if $dominated(e_c, SKY(q_i))$ OR $dominated(e_c, H_P)$ then
dominated := true; continue;
if $e_c$ is a non-leaf entry then
Expand $e_c$ for all relevant qi, insert children into $E_C(q_i)$;
else
foreach $e_p \in E_P(q_i)$ do
$midpoint(e_p, q_i) \rightarrow m$;
if $e_c$ is dominated by m then
if $e_p$ is a leaf entry then
if ($dominated(m, SKY(q_i))$ == false) then
$SKY(q_i)$. push(m);
$H_P.push(e_p)$;
dominated := true; break;
else
Expand $e_p$ for all relevant $q_i$, insert children into $E_P(q_i)$ ;
$E_P(q_i).remove(e_p)$;
if (dominated == false) then
$RSKY(q_i).push(e_c)$;
return $RSKY(q_i)$;
```

Points which are adjacent to the multi-dimensional space are more likely to benefit from parallel processing. For this reason the algorithm gERS originally by setting up Q on $[|Q|\ /\ G\ ]$ groups using a space filling curve (p, x, Hilbert curve). Then the algorithm processes the group's one after the other. For each group is selected at each iteration in a circular fashion (product candidate) (line 5 of Algorithm 6) and executed by a modified version of the ERS algorithm. Batch-ERS extends the ERS algorithm for the case of a group parallel processing queries. We then describe the differences in Batch-ERS algorithm with respect to ERS.

First, whenever a node e_x is accessed, the priority queues of all group members in which e_x is included are properly informed. Also, if a leaf node is found point pi (line 12 of the function 6) , the algorithm decides whether the pi should be inserted into a buffer H_P containing vantage points, i.e. those that can be used in pruning other candidates nodes. Intuitively, the closer is a candidate point, so having maximized opportunities for pruning a larger piece of the multidimensional dominated space. Following this logic, we implemented the buffer H_P as one key priority queue with the minimum Euclidean distance of a node from any candidate group. When the H_P is full the H_P is replaced with a new point p_i. The vantage points (corresponding essentially midpoints) are used in order to implement additional dominance check of all skyline (second condition - line 5) to avoid some unnecessary I/O operations.

6. EXPERIMENTAL EVALUATION

In this section we experimentally evaluate the proposed algorithms. All algorithms tested were implemented in C + +, compiled with gcc and executed on a system with processor 2 GHz Intel Xeon, memory RAM 4 GB.

6.1 Experimental Methodology

In our experiments we used a synthetic data generator designed to build datasets having different distributions with respect to their attribute values. Specifically, all uniform values are selected from a uniform distribution. Additionally, the attribute values were normalized to the range [0, 10000]. At the end we tag both data sets (products and consumer preferences) with the help of an R-tree considering the size of each page is equal to 4096 bytes.

In our experiments we compared the performance of the proposed algorithms ERS and gERS compared with the algorithm BRS, for each algorithm we measured the execution time and the number of I/O operations required to process (a) a set of |Q| reverse skyline queries, and (b) a query k-Dominant given a set of |Q| candidate products. More detail in each experiment was measured:

The number of I/O operations accessed from disk separately for products and consumer preferences. For each data set we used a buffer having a size equal to 100 pages (410 KB), which for our basic experiment represents 12.5% of the total data size. For buffer we followed the strategy of Least Recently Used cache replacement policy - LRU.

PROCESSING TIME: The total processing time consisted of the time spent in the CPU plus the cost of entry/exit from the disk, making the assumption that each input / output requires 1 millisecond.

Recalling that in the case of simple reverse skyline queries algorithms BRS and ERS will perform a set of queries serially one after the other. To evaluate the specific algorithms query k-Dominant is added with two additional steps of execution, (a) a preprocessing step which classifies points based on the hash function according to Hilbert curve, and (b) a final stage which implement the greedy algorithm kESA in order to choose k product candidates. In our experiments, the execution times of these two steps were negligible compared with the time required for the valuation of an inverse skyline query. Moreover it is worth noting that none of the algorithms were practically affected by the number of results k, as it should anyway be the first step to identify sets of influence for all Q product candidates.

In each experiment, we modify a single parameter while holding the other parameters at their default values, Table 1 shows the parameters concerned with the range of values tested for each parameter.

Table 1: Experimental parameters

Parameters	**Search range**
Number of dimensions (D)	2, 3, 4, 5
Size of dataset $P(\|P\|)$	10K, 100K, 500K, 1M
Size of dataset $C(\|C\|)$	10K, 100K, 500K, 1M
Size of cache / Total data size (M)	6.25%, 12.5%, 25%
No. of query per group (G)	5, 10, 20, 50, 100

6.2 Experimental results

Performance relative to the number of dimensions:

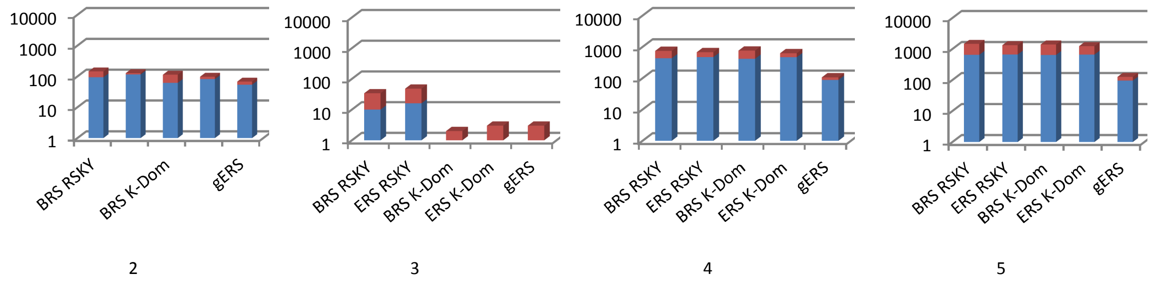

Figure 7 (a). No. of I/Os with respect to the number of dimensions

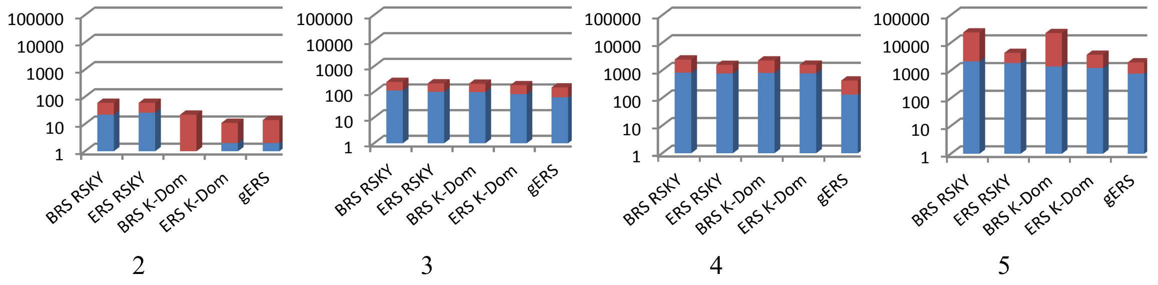

Figure 7 (b). Total processing cost in relation to the number of dimensions

In the first experiment we examine the performance of all algorithms as we vary the number of dimensions of 2-5, Figures 7 (a) -7 (b) show the number of I/O operations and the total processing time respectively. The algorithm BRS has prohibitively high cost of enforcement for at least three dimensions data, specific costs of 3.35 times more in CPU time in relation to the ERS algorithm even for two dimensions, and is approximately 46 times slower compared to the time spent in the CPU and 13.5 times slower compared to the total time for data 5D. It is worth mentioning that in our experiments we also tested even higher values for number of dimensions, e.g. for D = 6, which we have not included in the charts. In this case, the BRS algorithm took about 15 hours to terminate in our system, and the ERS algorithm took 20.2 minutes. Also in all experiments we observe that the algorithm gERS has much better performance compared to ERS and BRS in case of k-Dominant queries.

Another important observation we can make is that when we have a larger number of dimensions, the total processing cost is largely determined by the processing time of the CPU rather than the inputs/outputs of the disc. This is because the size of the global skyline SKY(L) and SKY(U) increased dramatically with the number of dimensions, therefore the number of dominance checks required to perform an reverse skyline query rises sharply . Note that the remaining part of our experimental evaluation we used the default value for the number of dimensions the value D = 3, which is relatively short for real data. Therefore, our experimental scenarios are rather favorably to competitive BRS algorithm. The efficiency gain of the ERS algorithm in respect to the BRS is significantly greater if we consider a larger number of dimensions.

Performance relative to the size of the dataset:

We then analyze experimentally the performance of the algorithms on the size of the dataset. Altering the original size of the dataset for products |P|. Figures 8 (a) -8 (b) show the results for the sets data. It is worth observing that the behavior between BRS and ERS algorithms with respect to the type of disk accesses shown in (Figure 8 (a)), BRS algorithm requires more disk accesses for data for all P, while the ERS algorithm performs more accesses to node q for all C, In this case the sizes of the sets P and C are similar (100K entries each). Both algorithms require approximately the same number of I/O operations. However, as the number of products increases, the strategy followed by the ERS algorithm is proved more effective in relation to the total number of I/Os required. Moreover, the time spent in the CPU of the ERS algorithm is considerably less and shows better scaling behavior as the size of the total P. Also, as shown in Figures, the algorithm gERS is by far the most efficient choice for k-Dominant queries, and seems not to be influenced particularly by varying size of the entire P.

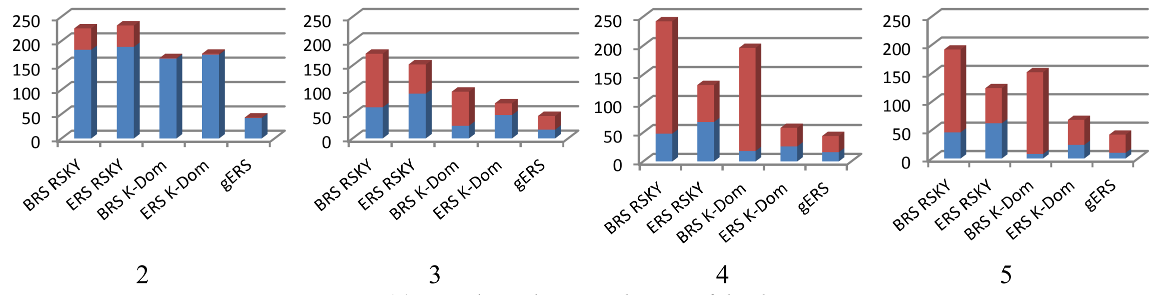

Figure 8(a). No. I/Os relative to the size of the data set P

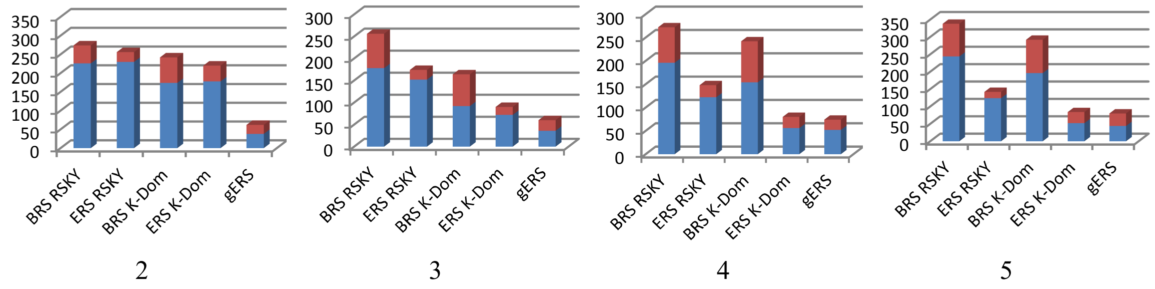

Figure 8(b). Total processing cost in relation to the size of the data set P

Then we compared the performance of all algorithms by varying the size of the total C. In Figures 9(a) -9(b) shows the number of I/Os and processing times for data sets, algorithms ERS and BRS require approximately the same number of I/O operations for larger datasets C. If the total size of C is much larger than the size of the total P, the strategy pursued by the algorithm BRS is more promising. However, as shown by experiments (Figures 9 (b)), the total processing cost of RSA algorithm is less than BRS algorithm mainly due to the significantly lower time spent in the CPU. Similarly, in this experiment, the algorithm gERS is significantly more efficient than the algorithms RSA and BRS in the case of execution of multiple reverse skyline queries.

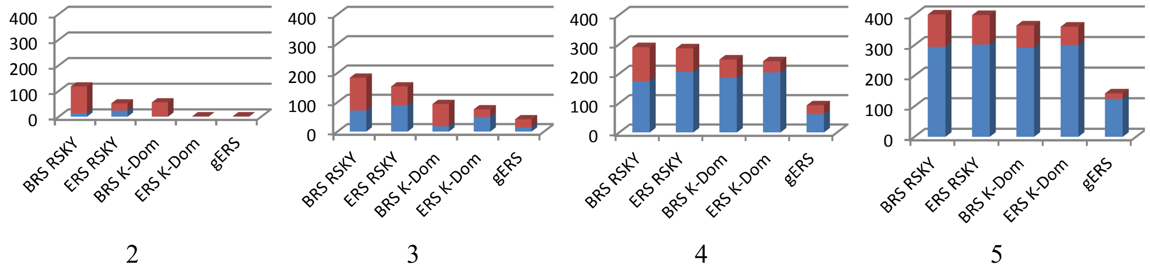

Figure 9(a). No. I/Os relative to the size of the data set C

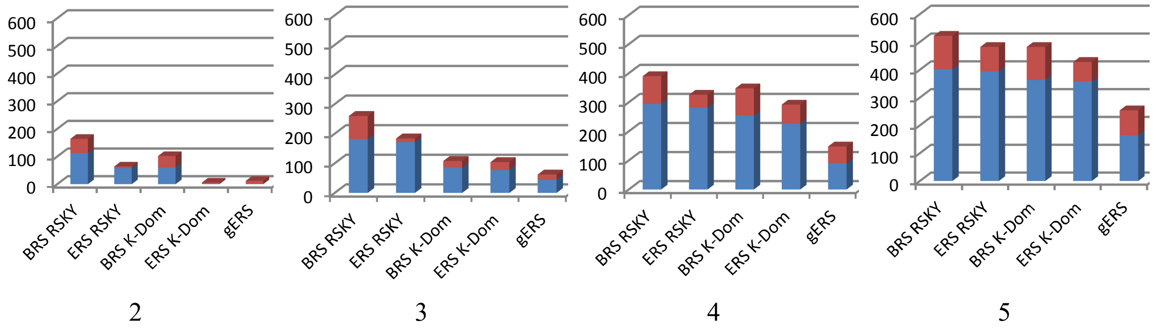

Figure 9(b). Total processing cost in relation to the size of the data set C

Performance relative to the number of queries per group

We evaluate experimentally the performance of the algorithm gERS relative to G the number of queries that are executed in parallel in a group and changing the value of G from 5 to 100 queries. Figures 10(a) -10(b) show the experimental results regarding the number of I/Os and the processing time in the CPU for data sets. As expected, the more the queries are executed in parallel, so the fewer disk accesses required, as one used to access the pruning nodes to many queries at once. However , as we can observe a number of queries over the total execution costs increased significantly due to higher time required for managing priority queues and hence the number of dominance checks needed . As shown in our experiments.

Progressive Productions result

In this experiment we examine the progressive production results of the algorithms ERS and BRS during the execution of a reverse skyline queries set |Q|. The x-axis represents the percentage of the results that have been determined in relation to the final influence score, The y axis shows the corresponding execution time required in absolute values (Figures 11(a)) and a percentage of total time of valuation queries (Figures 12 (b))for the datasets. Both figures show that the ERS algorithm shows much better behavior for the progressive production results in comparison with the BRS, particularly for determining the initial results. Specifically, the ERS algorithm returns 5% of the results in one tenth of the time required by the algorithm BRS. This advantage is especially important for applications that require fast response and do not need the full set of results.

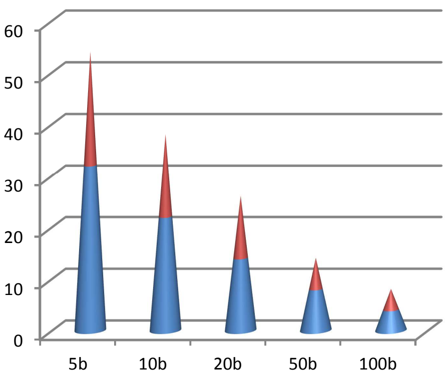

Figure 10(a). No. of I/Os in relation to the number of queries per group

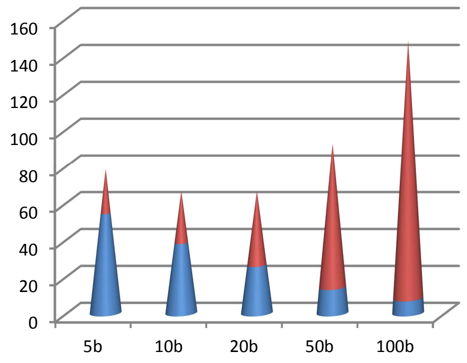

Figure 10(a). Total processing cost in relation to the number of queries per group

Figure 10. Performance relative to the number of queries per group

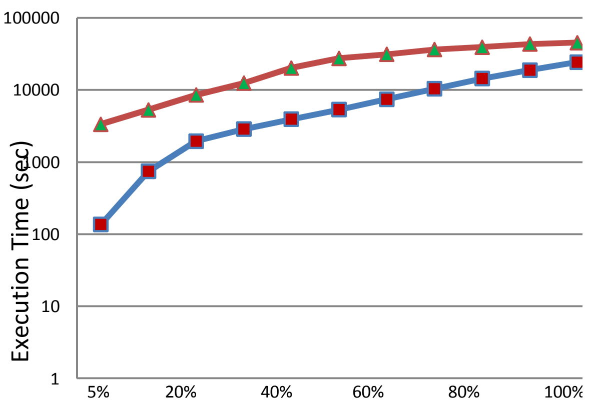

Figure 11(a). Processing time compared to the results obtained

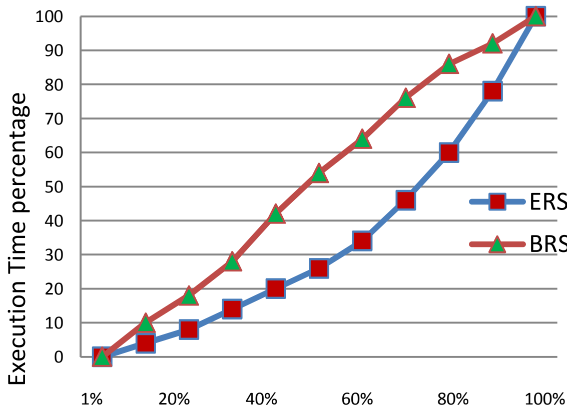

Figure 11(b). Processing time/Total time (%) compared to the results obtained

Figure 11. Progressive production results

7. CONCLUSION

In relation to market analysis using consumer preferences with an objective to effectively promote products and services: We developed new algorithms for two problems related to the analysis of large volumes of consumer preferences, with practical applications in market research. Moldings these two problems as variants of a multiple reverse skyline queries respectively. Firstly we proposed a new algorithm, called ERS for evaluating reverse skyline queries; the concluded experiments shows RSA algorithm significantly outperforms BRS in case of a reverse skyline query in relation to the speed of execution (performance), the scalability (scalability), and progressive production results (progressiveness), particularly for multidimensional data. Secondly we developed a variant of the ERS algorithm for groups of queries which significantly reduces the execution time required in relation to basic query execution by appropriate grouping similar products candidates, performing common accesses to disk, and allowing the simultaneous processing of multiple queries. Then we applied this new algorithm for evaluating k-Dominant queries. The experiment shows the algorithm we propose to simultaneously perform multiple queries outperforms methods that process each query individually.

REFRENCES:

[1]. Xiaobing Wu, Yufei Tao, Raymond Chi-Wing Wong, Ling Ding, and Jeffrey Xu Yu, *Finding the influence set through skylines*, EDBT, 2009, pp. 1030-1041.

[2]. Chen-Yi Lin, Jia-Ling Koh, and Arbee L.P. Chen, *Determining k-most demanding products with maximum expected number of total customers*, TKDE(2012).

[3]. Yu Peng, Raymond Chi-Wing Wong, and Qian Wan, *Finding top-k preferable products*, TKDE 24 (2012), no. 10, 1774-1788.

[4]. Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger, *Progressive sky-line computation in database systems*, TODS 30 (2005), no. 1, 41-82.

[5]. Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan, *A microeconomic view of data mining*, Journal of Data Mining and Knowledge Discovery 2 (1998), no. 4, 311-324.

[6]. Cuiping Li, Beng Chin Ooi, Anthony K. H. Tung, and Shan Wang, *Dada: a data cube for dominant relationship analysis*, SIGMOD, 2006, pp. 659-670.

[7]. Evangelos Dellis and Bernhard Seeger, *Efficient computation of reverse skyline queries*, VLDB, 2007, pp. 291-302.

[8]. Xiang Lian and Lei Chen, *Monochromatic and bichromatic reverse skyline search over uncertain databases*, SIGMOD, 2008, pp. 213-226.

[9]. Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Norvag, *Reverse top-k queries*, ICDE, 2010, pp. 365-376.

[10]. Prasad Deshpande and Deepak P, Efficient *reverse skyline retrieval with arbitrary non-metric similarity measures*, EDBT, 2011, pp. 319-330.

[11]. Thomas Bernecker, Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz,

Shiming Zhang, and Andreas Zufle, *Inverse queries for multi-dimensional spaces*, SSTD, 2011, pp. 330-347.

[12]. Muhammed Miah, Gautam Das, Vagelis Hristidis, and Heikki Mannila, *Standing out in a crowd: Selecting attributes for maximum visibility*, ICDE, 2008, pp. 356-365.

[13]. Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han, *Promotion analysis in multi-dimensional space*, PVLDB 2 (2009), no. 1, 109-120.

[14]. Tianyi Wu, Yizhou Sun, Cuiping Li, and Jiawei Han, *Region-based online pro-motion analysis*, EDBT, 2010, pp. 63-74.

[15]. Qian Wan, Raymond Chi-Wing Wong, Ihab F. Ilyas, M. Tamer Ozsu, and Yu Peng, *Creating competitive products*, PVLDB 2 (2009), no. 1, 898-909.

[16]. Akrivi Vlachou, Christos Doulkeridis, Kjetil Novag, and Yannis Kotidis, *Identifying the most influential data objects with reverse top-k queries*, PVLDB 3 (2010), no. 1, 364-372.

[17]. Qian Wan, Raymond Chi-Wing Wong, and Yu Peng, *Finding top-k profitable products*, ICDE, 2011, pp. 1055-1066.

[18]. Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei, *Distance-based representative skyline*, ICDE, 2009, pp. 892-903.

[19]. Guoren Wang, Junchang Xin, Lei Chen, and Yunhao Liu, *Energy-efficient reverse skyline query processing over wireless sensor networks*, TKDE 24 (2011), no. 7, 1259-1275.

[20]. Flip Korn and S. Muthukrishnan, *Influence sets based on reverse nearest neighbor queries*, SIGMOD, 2000, pp. 201-212.

[21]. Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du, *On computing top-t most influential spatial sites*, VLDB, 2005, pp. 946-957.

[22]. Jon Bentley, Kenneth Clarkson, and David Levine, *Fast linear expected-time algorithms for computing maxima and convex hulls*, SODA, 1990, pp. 179-187.