Under consideration for publication in Knowledge and Information Systems

Dominance Relationship Analysis with Budget Constraints

Shen Ge #1, Leong Hou U *2, Nikos Mamoulis #3, David W.L. Cheung #4

#Department of Computer Science, University of Hong Kong
Pokfulam Road, Hong Kong
[1]sge@cs.hku.hk [3]nikos@cs.hku.hk [4]dcheung@cs.hku.hk

*Department of Computer and Information Science, University of Macau
Av. Padre Tomás Pereira Taipa, Macau
[2]ryanlhu@umac.mo

Abstract. Creating a new product which dominates all its competitors is one of the main objectives in marketing. Nevertheless, this might not be feasible since in practice the development process is confined by some constraints, e.g., limited funding or low target selling price. We model these constraints by a constraint function which determines the feasible characteristics of a new product. Given such a budget, our task is to decide the best possible features of the new product that maximize its profitability. In general, a product is marketable if it dominates a large set of existing products, while it is not dominated by many. Based on this, we define dominance relationship analysis (DRA) and use it to measure the profitability of the new product. The decision problem is then modeled as a *budget constrained optimization query* (BOQ). Computing BOQ is challenging due to the exponential increase of the search space with dimensionality. We propose a divide-and-conquer based framework which outperforms a baseline approach in terms of not only execution time but also space complexity. Based on the proposed framework, we further study an approximation solution which provides a good tradeoff between computation cost and quality of result.

Keywords: dominance relationship analysis, budget constrained optimization query

1. Introduction

Given a set of existing products $\mathcal{O}$ in the market and a development budget B, our task is to decide the best possible features of a new product that maximize its profitability

Received Apr 27, 2012
Revised May 15, 2013
Accepted Aug 24, 2013

and fit the budget B. For example, consider the development of a new laptop computer by a manufacturer. Typically, *performance* and *weight* are two primary features that customers care when choosing a laptop. Assuming that the selling price is constrained, the manufacturer may find it hard to create a new laptop that maximizes the quality of both features at the same time. A feasible solution may either have to sacrifice performance (e.g., using a low-end processor) or to reduce weight (e.g., replacing a 6-cells by a 4-cells battery), in order to fit the target price.

Web advertising is another example. A provider wants to create a new advertisement package that fits the typical service price and also make the package attractive to the potential customers. A customer may consider the following features for her ad: the position of the ad in the web page and the length of display time. An ad is more expensive if it is displayed at a good position for a long time. Assuming that the service provider has a targeted service price, our task is to create a new package that dominates most existing products in the market. Once again, the problem is to find the optimal feature values, constrained by a budget (the service price).

Motivated by these examples, we study an optimization query, called *budget constrained optimization query (BOQ)* in this paper, defined as Problem 1 below. Typically, manufacturers do not have infinite development funding and must consider different trade-offs and constraints when they do research and development for their products. Given a set of products $\mathcal{O}$ in the market and a budget B, the goal of BOQ is to identify the features of a new product x such that x fits the budget B and its profitability is maximized.

Problem 1 (Budget Constrained Optimization). Given a budget B, a constraint function $C(x)$, and an objective function $f(x)$, create a new product x such that $C(x) \leq B$ and the objective function $f(x)$ is maximized.

Constraint function $C(x)$. We consider constraint functions $C(x)$, which monotonically change with the values of the features (i.e., dimensions) of x. This implies that for $C(x)$ to remain constant, if a feature value increases, there should be another feature of decreasing value. The weights of different features in the function could differ and the function is not necessarily a linear combination of the features. In this work, $C(x)$ is modeled by a set of monotone functions [1], which are highly flexible and widely used in many works.

$C(x)$ is a generalized concept and is defined by domain experts in general. For instance, the value of an NBA player can be calculated by the approximate value function (AV function) given in (Oliver, 2010). The AV function is monotonic to the players' performance, increasing with all positive factors (points, rebounds, etc.) and decreasing with all negative factors (field goals missed, turnovers, etc.). The AV function is an example of a constraint function in a real application that measures the value of objects based on their capabilities.

We use NBA player statistics to demonstrate the relationship between players capacities (*profitability*) and salaries (*cost*). We plot the average points and assists of guard players in 2007-2008, in data collected from (*NBA Basketball Statistics*, 2009). These two are the main features that characterize the capabilities of a guard. We also use salary information from (*Pro Basketball in USAToday*, 2009), to illustrate the top 80 paid guard players in Figure 1. Each player is displayed as a point on the points-assists plane, while different colors and markers are used to distinguish players with different salary scales.

[1] $C(x)$ can be a piecewise function. The solution of piecewise constraint functions is discussed in Appendix B.

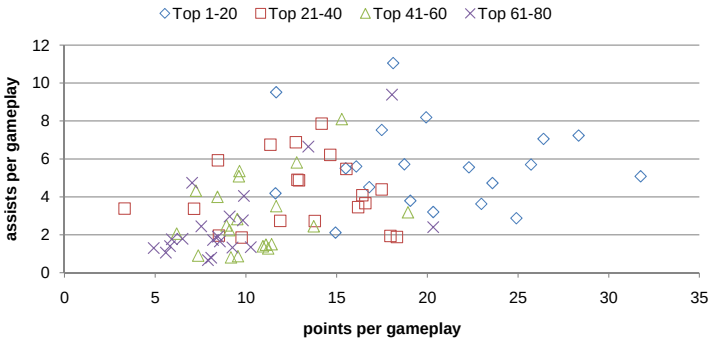


Fig. 1. Average points and assists performance of the top paid guard players in NBA

The figure shows the general trend that, the higher the salary, the better the capabilities. This coincides with our intuition that if our budget is large, our space of choice becomes larger, and thus we can find better players. Also notice that, for the same salary level, there are tradeoffs between the two capability values, which prevent us from maximizing both two values at the same time under a budget constraint.

Objective function $f(x)$. $f(x)$ is the second and equally important component of BOQ, used to measure the quality of a product based on the concept of dominance, as shown in Definition 1.

Definition 1 (Dominance). A product x is definitely better than (*dominates*, $\prec$) another product x' if and only if every feature of x is not worse than the corresponding feature of x' and there is at least one feature where x is strictly better.

In general, a product is marketable if it *dominates* a large set of existing products and it is *not dominated* by many. (Li, Ooi, Tung and Wang, 2006; Zhu, Li, Tung and Wang, 2012) was the first work to propose a definition for *profitability* based on dominance relationship analysis (DRA), where DRA is a well accepted model in multidimensional analysis. (Li, Tung, Jin and Ester, 2007; Papadopoulos, Lyritsis, Nanopoulos and Manolopoulos, 2007; Yang, Wang and Kitsuregawa, 2007; Yang, Li and Kitsuregawa, 2008). In this work, the objective function $f(x)$ is defined as follows:

$$f(x) = (1+\beta)dc^{+}(x) - (1-\beta)dc^{-}(x) \tag{1}$$

In Eq. 1, $dc^{+}(x)$ is the number of objects being dominated by x, $dc^{-}(x)$ is the number of objects dominating x, and β is a weighting parameter adjusting the relative weights between $dc^{+}(x)$ and $dc^{-}(x)$, which is set to 0 by default, giving an equal weighting to both sides. Actually, as we can see later, β can take any value in the range $[-1, 1]$ without affecting the correctness of our algorithms. A large value of $f(x)$ implies a product x, which is better than a large set of competitors while being worse to only few of them. If there was no budget constraint, we could develop a product with the best possible values in all features, which would dominate all products in the market, while being dominated by none of them; thus, the optimization problem is meaningless without the constraint $C(x) \leq B$.

In the rest of the paper, we make the convention that *smaller* feature values are *better*. To be consistent with this assumption, we also assume that the constraint function is antimonotonic to the feature values; that is, if $x[i] \leq y[i]$ in every dimension i, then

Table 1. The features and price of six laptop models

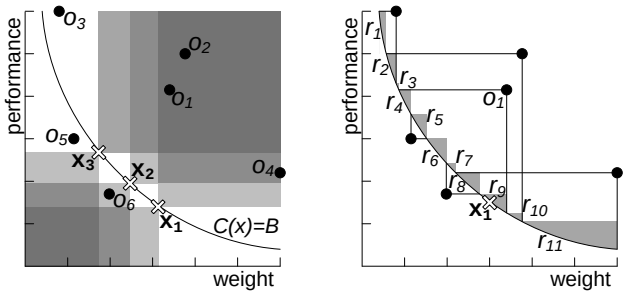
	Laptop model	CPU (rank)	Weight	Price
o_1	Samsung SF410-A01	i3-380M (303)	4.82 lbs	$499.99
o_2	HP 630 LV970UT	P6200 (446)	5.5 lbs	$349.99
o_3	HP Mini 5103	N455 (1036)	2.64 lbs	$719.99
o_4	Lenovo G770	i5-2410M (201)	6.61 lbs	$629.99
o_5	Samsung NP900X3A	i5-2467M (280)	2.88 lbs	$1,599.99
o_6	Sony VPC-SA2HGX	i7-2620M (141)	3.7 lbs	$1,869.99

$C(x) \geq C(y)$. Table 1 lists six laptop models and two features of them: CPU rank[2] and weight. In addition, their selling prices[3] are shown. Figure 2 plots the notebooks as points in the two-dimensional CPU-weight space. Typically, a laptop with higher price has better specifications (i.e., better performance and lighter weight). o_6 is a highly-rated laptop based on our objective function $f(x)$, since it dominates 3 other models (i.e., o_1, o_2, and o_4) but is not dominated by any. Nevertheless, o_6 has the highest cost $C(x)$ (i.e., highest selling price) among all these models. When designing new models, manufacturers are constrained by cost. For example, the feasible laptops that can be designed, given a target selling price B=$1,000, could be modeled by the points above the curve $C(x) = B$ in Figure 2(a). Given the set of existing notebooks $\mathcal{O}$, constraint function $C(x)$, and a development budget B=$1,000, as shown in the figure, our task is to determine the best possible features of the new product x such that $C(x) \leq B$ and $f(x) = dc^+(x) - dc^-(x)$ is maximized (assuming smaller values in each dimension are better). Given three candidates x_1, x_2, and x_3, their profitabilities calculated by $f(x)$ are 3, 2, and 2 respectively; thus, x_1 is the best over these candidates. In Figure 2(b), we show 11 *feasible profitable regions*, shaded in gray, on top of the budget plane $C(x) = B$. These regions are defined by the points on the line $C(x) = B$, where the line intersects the hyperplanes orthogonal to existing products (indicated by black dots). After the partitioning, all points within each region have identical profitabilities, while satisfying the objective function. For each region, we can easily measure the profitability by counting the products it dominates / it is dominated by. For example, the region with the highest profitability is r_9; $\forall x \in r_9, f(x) = 3$ (e.g., $f(x_1) = 3$).

In general, it is more desirable to identify the most profitable regions (as in Figure 2(b)), instead of simply comparing some random candidate products (as in Figure 2(a)). First, this offers flexibility to the developer to choose from a range of possible feature value combinations that fall in the most profitable regions. Second, she might minimize the production cost (while maximizing the profit) by choosing the point x in the most profitable regions, which has the lowest $C(x)$ value (i.e., the upper right corner of r_9 in our example). Therefore, we define a generalization of BOQ (GenBOQ), which returns a set of regions that share the best profitability instead of a single object in BOQ.

Problem 2 (General BOQ). Given a budget B, a constraint function $C(x)$, and an objective function $f(x)$, a general BOQ returns a set of regions R such that $C(x) \leq B$ and the objective function $f(x)$ is maximized, $\forall x \in R$.

[2] based on http://www.cpubenchmark.net/cpu_list.php (Nov 2011)
[3] collected from http://www.compusa.com (Nov 2011)



(a) Computation of BOQ (b) Profitable Regions

Fig. 2. Examples of BOQ

The big challenge of the query is the huge solution search space. Instead of verifying infinite potential products on the continuous budget plane, a possible (baseline) approach is to generate compact regions by drawing orthogonal hyperplanes from each existing object (as shown in Figure 2(b)). However, as we show in the following lemma, the number of compact regions is $O(d \cdot n^{d-1})$, in a d-dimensional problem with n existing products.

Lemma 1 (Number of compact regions). The number of compact regions that can be generated in a d-dimensional problem with n objects is $O(d \cdot n^{d-1})$.

Proof. Suppose there are n discrete objects in d dimensional space, the entire space can be split by these n objects into $O(n^d)$ regions (i.e., MBRs). The problem is now to find how many these $O(n^d)$ regions intersect the budget plane $\mathcal{BP}$, in the worst case. The largest $\mathcal{BP}$ can be constructed by d orthogonal hyperplanes, each of which is orthogonal to one axis i and its i-th feature value is set to the minimum value. The intersection point of all hyperplanes is the minimum corner in the entire space. Therefore, each hyperplane intersects $O(n^{d-1})$ regions and we have d hyperplanes; the total number of compact regions is $O(d \cdot n^{d-1})$. □

To address this space growth challenge, we develop an efficient divide-and-conquer framework that partitions the space recursively. Our approach greatly outperforms the baseline method in terms of response time. In addition, we reduce the memory requirements from $O(d \cdot n^{d-1})$ to $O(\theta^{d-1})$, where θ is a constant that expresses a tradeoff between computational cost and space.

Furthermore, we study an approximation solution of GenBOQ (ApprGenBOQ) which returns only one feasible region r_{appr} such that the profitability of r_{appr} is bounded by a tolerance T from the optimal. ApprGenBOQ can find a result efficiently without sacrificing much precision (which is controlled by T). Formally, we define ApprGenBOQ as follows.

Problem 3 (Approximation of GenBOQ). (APPRGENBOQ) Consider a budget B, a constraint function $C(x)$, and an objective function $f(x)$. Let R be the (optimal) regions returned by GenBOQ. ApprGenBOQ returns any region r_{appr} such that, $\forall x \in R, \forall y \in r_{appr}$, we have $|f(x) - f(y)| \leq T$ where T is a given quality tolerance bound.

The rest of the paper is organized as follows. Section 2 presents related work. Pre-

liminary concepts and properties are introduced in Section 3. In Sections 4 and 5 we present our solution and Section 6 evaluates it. The paper is concluded in Section 7.

2. Related Work

In this section, we discuss work related to multidimensional data analysis based on dominance relationships between objects, skyline queries, which again use the definition of object dominance to search the multi-dimensional space, and additional related queries.

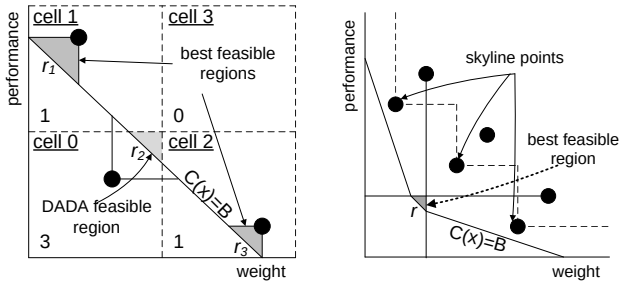
2.1. Dominance Relationship Analysis

Ref. (Li et al., 2006) was the first paper for business analysis from a microeconomic perspective using the concept of dominance. The authors proposed a data cube model (DADA), to summarize all the domination relationships between objects in all dimensions, in a bottom-up fashion. In DADA, the space is divided into $Dom_1 \times Dom_2 \times \ldots \times Dom_d$ cells, where Dom_i is the number of discrete values stored in dimension i. The objects inside the same cell have identical domination relationship which can be used to save the computation complexity. Furthermore, the authors proposed a D*-tree to group all neighboring cells with the same domination count to support more efficient searching. To address the microeconomic problems, they proposed three queries, including Linear Optimization Query (LOQ), Subspace Analysis Query (SAQ) and Comparative Dominant Query (CDQ), which can be computed efficiently using the DADA model.

The definition of our GenBOQ is similar to LOQ proposed in (Li et al., 2006; Zhu et al., 2012). However, our problem has two main differences: (i) Our GenBOQ supports any combination of monotone functions while LOQ merely models the budget constraint as a single linear function. (ii) DADA can only return a set of *cells* that maximize the objective function instead of compact regions as proposed in our work. We demonstrate difference (ii) using Figure 3(a). Suppose that there are three objects in the two dimensional space and DADA regularly divides the space into 4 cells. The domination scores $f(x)$ are shown in the left bottom of the cells. DADA will return cell 0 (equiv. to r_2) for LOQ since the cell has the maximum domination score in all four possible cells. Nevertheless, r_2 is not the best region since its domination score $f(r_2)$ is -1, which is worse than $f(r_1)$ and $f(r_3)$. Note that if the features values are discrete, DADA can return the correct result for LOQ. However, the computation becomes prohibitive if the domain of discrete values is large.

Yang et al. in (Yang et al., 2008) and (Yang et al., 2007) study some extensions of DRA. In (Yang et al., 2007), a special data cube called *ParCube* that supports dominance relationships analysis on partially ordered dimensions is proposed. They proposed algorithms for updating their index structure and return the exact objects dominated rather than just their total number (as in (Li et al., 2006; Zhu et al., 2012)). Ref. (Yang et al., 2008) relaxes the dominance relationship and compresses the index with such dominance relationships encoded; in addition, it proposes some efficient strategies to support querying on relaxed dominant relationships. However, all these methods are built on the data cube concept and none of them can compute the exact result of our optimization problem.

Domination game analysis for microeconomic data mining is studied in (Zhang, Lakshmanan and Tung, 2009). Given a set of customers and a set of manufacturers, each manufacturer creates only one product to the market for fairness. The task of the domination game is to find a configuration of the products that achieves stable expected



(a) with DADA (b) with skyline

Fig. 3. Comparison of GenBOQ with other queries.

market shares for all products. A product is said to dominate a customer if all its features can satisfy the requirements of the customer. The expected market share of a product is measured by the expected number of buyers in the customers, all of which are equally likely to buy any product dominating them. A depth-first and a breadth-first search strategy are proposed along with a set of pruning techniques for the search space. The techniques proposed in (Zhang, Lakshmanan and Tung, 2009) cannot be used for our problem, because they find a configuration rather than a set of regions that maximize an objective function.

2.2. Skyline Queries

There is plenty of work on skyline evaluation (e.g., (Börzsönyi, Kossmann and Stocker, 2001; Tan, Eng and Ooi, 2001; Kossmann, Ramsak and Rost, 2002; Chomicki, Godfrey, Gryz and Liang, 2003; Papadias, Tao, Fu and Seeger, 2005; Zhang, Mamoulis and Cheung, 2009)). The concept of skyline is based on the dominance relationship and originated from maximal vector problem (Kung, Luccio and Preparata, 1975).

The objective of skyline queries is to find the objects that are not dominated by others. They were first introduced in a database context in (Börzsönyi et al., 2001). In (Börzsönyi et al., 2001), approaches using block nested loops (BNL) and divide-and-conquer (D&C) were proposed for skyline computation. Other well known methods include the bitmap method (Tan et al., 2001), sort-first-skyline (SFS) (Chomicki et al., 2003), and branch and bound skyline (BBS) (Papadias et al., 2005). BBS (Papadias et al., 2005) is an incremental skyline algorithm that accesses a minimal number of nodes from an R-tree (Beckmann, Kriegel, Schneider and Seeger, 1990) that indexes the data. An object-based space partitioning method that provides efficient skyline computation in high dimensional spaces was proposed in (Zhang, Mamoulis and Cheung, 2009).

The top-k dominating query, studied in (Yiu and Mamoulis, 2009) returns k data objects that dominate the largest number of objects in the dataset. The main differences to our work are that this method searches in the much smaller search space of the data objects to find the best one and we consider both dominating and dominated objects. In fact, no skyline method can be adjusted to solve our problem. This is because our query aims at finding compact regions instead of existing objects, and these regions are much more ($O(d \cdot n^{d-1})$ according to Lemma 1) than the original n objects. Also the

final best compact regions are not necessarily defined by skyline points. For example, in Figure 3(b), the gray area denotes the best feasible region r. As we can see, r cannot be constructed even if we know all skyline points. Obviously, the skyline operator cannot be used to solve our GenBOQ problem.

2.3. Related Queries

A variant of the top-k dominating queries (Yiu and Mamoulis, 2009), which returns the k most demanding products instead of the top-k dominating products, is recently studied in (Lin, Koh and Chen, 2012). The demanding score of a product is defined as the expected number of customers who are willing to buy this product. This problem is shown to be NP-hard when the product feature dimensionality becomes 3 or higher. The authors propose an upper bound pruning algorithm for exact solution as well as a greedy approximation method, which were shown to have comparable performance.

Miah et al. (Miah, Das, Hristidis and Mannila, 2008) studied an optimization problem that selects a subset of attributes of a product t such that t's shortened version maximizes t's visibility compared to other products to potential customers. This problem is NP-hard; several exact and approximate algorithms are proposed. This problem has a different definition compared to our problem, but shares the intuition that manufacturers want to cut down a development cost.

In (Wu, Xin, Mei and Han, 2009; Peng, Wong and Wan, 2012), the authors studied a problem that finds a subset of object o's features so that o is ranked highly in the found subspace. They call this *promotion analysis through ranking*; this is a challenging problem due to the explosion of the search space and the high aggregation cost. They proposed a *PromoRank* framework, and an efficient algorithm using subspace pruning, object pruning, and promotion cube. Since the aim is to find the promotive subspace for some specified objects, this framework cannot suggest possible values for newly developed products. Also, it does not consider budget constraints.

Wu et al. (Wu, Sun, Li and Han, 2010) study the search of the top-k most interesting regions for object promotion; an object is worth to be promoted in a specific feature region if it is highly ranked in that region. This work shares the same intuition with ours that discovering specific feature regions is important in marketing promotion. However, the solution of (Wu et al., 2010) is inapplicable to our problem as it applies on a different problem definition; in addition, it only discovers the specific feature regions approximately.

Creating competitive products from a pool of possible dimensional values is studied in (Wan, Wong, Ilyas, Özsu and Peng, 2009). The objective is to find a set of newly created objects according to some generating rules, which cannot be dominated by any existing products in the market. Instead of generating and checking all possible new objects, (Wan et al., 2009) uses group partitioning with partial pruning so that only a small subset of the new possible objects are considered. The idea of finding good possibilities for new product is basically the same with ours. However, we can suggest regions for new products even without knowing how these new products can be generated. Also, as opposed to (Wan et al., 2009), we consider budget constraints.

Wan et al. (Wan, Wong and Peng, 2011) suggest the identification of top-k profitable products from a set of new products P_{new}. For each product, all the features other than *price* are known. The problem is to assign appropriate prices to k products from P_{new} such that these products are not dominated by other existing products in the market. This problem is shown to be NP-hard when there are 2 or more dimensions in the feature space and can be approximately solved by greedy methods. We share the same intuition

as this work, which also creates new products that maximize an objective confined by constraints. Their objective is to create k new skyline products, while our objective is to find candidate products having the best domination score. Their constraint is the set of possible new products, while our constraint is a value of function $C(x)$.

Zhang et al. (Zhang, Jia and Jin, 2011) investigate a problem to identify promotional subspace in multi-dimensional databases. Other than the data cube techniques used in (Wu et al., 2009; Peng et al., 2012; Zhu et al., 2012), their solution exhaustively searches the promotional subspace combinations (i.e., dimension combinations) and terminate a search branch when it hits the early stop criterion. Their objective is quite different from ours as their solution reports a promotional subspace instead of a profitable region in this work.

A recent work (Lu and Jensen, 2012) studies a top-k product upgrading problem: the objective is to upgrade k existing products to be competitive in the market by the most economical way. A product is competitive if it is a skyline product in the current market. Accordingly, to make an existing product competitive, we can upgrade some of its features such that it is no longer dominated by any other products. The upgrading cost function in this work shares the same intuition with our constraint function $C(x)$.

2.4. Budget Optimization in Other Contexts

There are budget optimization problem in different contexts. (Zhou, Chakrabarty and Lukose, 2008) considers a budget-constrained bidding optimization problem for sponsored search auctions, and models it as an *online multiple-choice knapsack* problem. For a time period $t \in [1, T]$, if the profits and weights are $(v(t), w(t))$, and we want to maximize $\sum v(t)$ satisfying budget constraint $\sum w(t) \leq B$. Suppose $L \leq \frac{v(t)}{w(t)} \leq U$, the authors propose both deterministic and randomized algorithm with competitive ratio $\ln(U/L) + 1$ and a $(\ln(U/L) + 2)$ algorithm for multiple-choice knapsack problem. Their strategy can be summarized in one sentence: at any time t, if the fraction of budget spent is $z(t)$, bid $V/\Phi(z(t))$, where V is the expected value-per-click of the keyword, and $\Phi(t)$ is a continuous function of z which can be determined. Since we share no common aspect with this problem, such simple bidding strategy cannot help us to find the best place in feature space to maximize our dominance based profit function.

(Feng, Song, Zheng and Xia, 2003) studies a problem of scheduling dependent tasks in grid computing, given deadline and budget constraint. The authors model and solve their problem using binary integer programming. Since our target function is based on dominance count, which is discrete and cannot be captured by a simple set of equations or inequalities, we cannot use their binary integer programming techniques in our problem.

(Pujowidianto, Lee, Chen and Yap, 2009) investigates a problem of finding the best designs. The designs are evaluated by some measures which can be estimated by a serie of stochastic experiments. The aim of the problem is to maximize the probability of selecting the best design while obeying a computing budget. In our problem, we do not have stochastic experiments and our budget is an area constraint instead of a simple amount limit. As a result, we are not able to adopt any techniques from aforementioned papers.

In our problem, the budget constraint is nowhere same with the budget constraint in operations research. Typical budget constraint in operations research problem is a simple value threshold, defining the maximal total amount that we can spend. However, our budget constraint represents a continuous range, limiting the area that we can achieve

in the original product feature space. Since our problem shows no resemblance with previous budget constrained problem in operations research, we are not able to adapt their algorithms to our problem.

3. Preliminaries

In this section, we define concepts and present properties that are used in our solutions. We assume a dataset $\mathcal{O}$ of n data objects $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ in a d dimensional space. Each object $o_i = (o_i[1], \ldots, o_i[d])$ models the feature vector of an existing product in the market. We assume that for each feature, smaller values indicate better quality. Table 2 summarizes the notations that will be used throughout the paper. Given a constraint function C and a development budget B, we define the budget plane as follows.

Definition 2 (Budget plane). The budget plane $\mathcal{BP}$ is a $d-1$ dimensional surface, on which all points have the same development budget B. Formally, $\mathcal{BP}$ is the area defined by $C(x) = B$.

Definition 3 (Positive/Negative object set). Set $\mathcal{O}^+$ ($\mathcal{O}^-$) denotes the objects x in $\mathcal{O}$ for which $C(x) < B$ ($C(x) \geq B$).

The objects are classified into two categories, positive object set $\mathcal{O}^+$ and negative object set $\mathcal{O}^-$, based on Definition 3. Broadly speaking, an object is in $\mathcal{O}^+$ if it can add to the profitability of the new product. For instance, $\mathcal{O}^+ = \{o_1, o_2, o_3, o_4\}$ and $\mathcal{O}^- = \{o_5, o_6\}$ in Figure 2(a).

Definition 4 (Dominance set). $D^+(x)$ denotes the set of objects in O^+ which are dominated by object x. Similarly, $D^-(x)$ denotes the set of objects in O^- dominating x.

After classifying the objects, we can define the set of objects that dominate or are dominated by a given object x in Definition 4. In Section 1, we defined $dc^+(x)$ ($dc^-(x)$) as the number of objects that are dominated by (dominate) x. We have $dc^+(x) = |D^+(x)|$ and $dc^-(x) = |D^-(x)|$.

A minimum bounding rectangle (MBR) is used to approximate a region on the budget plane $\mathcal{BP}$ as shown in Figure 4. For instance, Figure 4(a) shows 11 MBRs $\mathcal{M} := \{m_1, ..., m_{11}\}$ shaded in gray. These 11 MBRs cover the entire budget plane of the example in Figure 2. Moreover, in this example, these 11 MBRs are the *finest* MBRs, according to Definition 5 below.

Definition 5 (Finest MBR). An MBR m is finest if and only if every object inside it shares the same profitability: $f(x) = f(x'), \forall x, x' \in m$.

We can split each MBR (finest or not), into two regions, using $\mathcal{BP}$. One of them, called the *feasible region* contains the points that satisfy the budget constraint. For example, r_9 in Figure 2(b) is the feasible region for MBR m_9 in Figure 4(a).

Definition 6 (Feasible MBR region). The feasible region of MBR m, denoted by $R(m)$ consists of all points x in m for which $C(x) \leq B$.

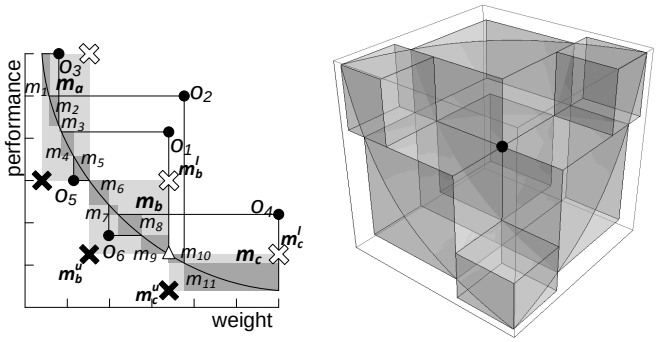
In general, we can find a set of MBRs which cover the entire budget plane, e.g., the lightgray MBRs $\{m_a, m_b, m_c\}$ in Figure 4(a). However, one or more MBRs may not be finest, which means that the points in them do not share the same profitability. For instance, the right-top corner m_b^l of MBR m_b, which is the point in it with the

Table 2. Summary of Notations

Symbol	Meaning
B	the development budget
$C(x)$	the constraint function
$f(x)$	the objective function
β	weighting parameter for $f(x)$
$\mathcal{BP}$	budget plane which is defined by $C(x) = B$
$\mathcal{O}$	the set of objects
o_i	an object in $\mathcal{O}$
n, $\|\mathcal{O}\|$	the number of objects
$\mathcal{O}^+(\mathcal{O}^-)$	objects $x \in \mathcal{O}$ where $C(x) < B$ ($C(x) \geq B$)
$D^+(x)$ ($D^-(x)$)	objects in $\mathcal{O}^+$ ($\mathcal{O}^-$) that are dominated by (dominate) x
$D^{\Delta+}(x)$ ($D^{\Delta-}(x)$)	objects in $\mathcal{O}^+$ ($\mathcal{O}^-$) that are strictly dominated by (strictly dominate) x
$dc^+(x)$ ($dc^-(x)$)	$\|D^+(x)\|$ ($\|D^-(x)\|$)
$dc^{\Delta+}(x)$ ($dc^{\Delta-}(x)$)	$\|D^{\Delta+}(x)\|$ ($\|D^{\Delta-}(x)\|$)
m, $\mathcal{M}$	an MBR m and the set of MBRs $\mathcal{M}$
$R(m)$	feasible region of an MBR m
m^l, m^u	right-top and left-bottom corner of MBR m
$f^l(m)$, $f^u(m)$	tight lower and upper profitability bound of m
$m.b^l$, $m.b^u$	lower and upper profitability bound of m
T	quality tolerance bound
γ	pruning bound for MBR refinement
η	pruning bound for approximated MBR refinement

largest coordinates in all dimensions, is the lowest profitable point in m_b. The left-bottom corner m_b^u is the highest profitable point. Thus, for an MBR m, we can simply bound the profitability of any point in m by $[f(m^l), f(m^u)]$, which is the bounds of area *inside* the MBR.

In order to get the tightest upper and lower bounds, we refine the profitability bound for a MBR by disregarding boundary cases. We note that the lower-left corner m_c^u of MBR m_c dominates (i.e., determines the profitability) all objects on the projected orthogonal line/hyperplane from m_c^u. However, we do not consider the boundary cases (e.g., o_1) in $f^u(m_c)$ since o_1 is not dominated by any objects *inside* m_c. More specifically, when calculating the profitability bound for a given MBR m, we consider only



(a) 2D MBRs (b) 3D MBRs

Fig. 4. Example of MBRs

strictly dominated objects in the upper bound, making the bound tighter. The strict dominance relationship is defined in Definition 7.

Definition 7 (Strict dominance set). $D^{\Delta+}(x)$ denotes the set of objects in O^+ that are strictly dominated by object x. Object x strictly dominates x' if and only if every feature of x is strictly better than the corresponding feature of x'. Similarly, $D^{\Delta-}(x)$ denotes the set of objects in O^- strictly dominating x.

A tight upper profitability bound $f^u(m)$ for MBR m is computed by ignoring points that are not strictly dominated by the upper bound corner m^u. Similarly, a tight lower bound $f^l(m)$ is computed by counting those objects that strictly dominate m^l. These bounds are derived from Eq. 1 (i.e., the definition of profitability).

$$f^u(m) := (1+\beta)dc^{\Delta+}(m^u) - (1-\beta)dc^{-}(m^u) \tag{2}$$

$$f^l(m) := (1+\beta)dc^{+}(m^l) - (1-\beta)dc^{\Delta-}(m^l) \tag{3}$$

$dc^{\Delta+}(m^u)$ ($dc^{\Delta-}(m^l)$) is the number of objects in $D^{\Delta+}(m^u)$ ($D^{\Delta-}(m^l)$). For instance, $[f^l(m_c), f^u(m_c)] = [1, 2]$ is the tightest profitable bound of m_c, and objects o_2 and o_4 are in $D^{\Delta+}(m_c^u)$. Finally, we can determine whether an MBR is finest, using the following property.

Property 1. MBR m is *finest* if and only if $f^u(m) = f^l(m)$.

The number of finest MBRs that cover the entire budget plane $\mathcal{BP}$ is $O(d \cdot n^{d-1})$, for n d-dimensional objects, as we show in Lemma 1. Figure 4(b) shows a 3D example. The entire $\mathcal{BP}$ is covered by 7 finest MBRs even though there is only one object, shown as a black point, in the domain space. In summary, finding the set of finest MBRs covering the entire $\mathcal{BP}$ is a hard problem.

4. Evaluation of GenBOQ

In this section we provide a comprehensive study on the evaluation of GenBOQ. First, we discuss the MBR refinement algorithm, which serves as a baseline approach for our problem. Then, we outline a divide-and-conquer approach paired with some optimization techniques. In addition, we describe two alternative approaches, including a

best-first search method with limited applicability and a hybrid approach that combines divide-and-conquer with best-first search. Finally, we analyze the complexity of MBR refinement algorithm and D&C framework.

4.1. MBR Refinement Algorithm

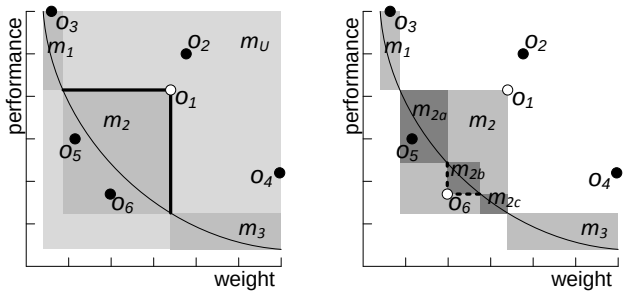
As discussed, every object inside the same finest MBR shares the same profitability. Assuming that we have a budget plane $\mathcal{BP}$ and a set of finest MBRs, the result of GenBOQ is the feasible region of one or more of these MBRs. For example in Figure 4(a), m_9 is the finest MBR having the maximum profitability and the feasible region of m_9 (shown as r_9 in Figure 2(b)) is the result of the query.

Therefore, our problem is reduced to finding the set of finest MBRs with maximum profitability. We propose an algorithm which *refines* the bounds of MBRs iteratively by accessing the objects. We use $m.b^l$ ($m.b^u$) to denote the general lower (upper) profitability bound (not necessarily tight) of MBR m. At each step, one or more MBRs m are refined into a set of smaller MBRs; the smaller MBRs have tighter profitability bounds. This process is demonstrated in Figure 5. Assuming that we only know how many objects are above/below the budget plane $\mathcal{BP}$, but not their locations, we can first define an MBR m_U which covers the entire $\mathcal{BP}$, as shown in Figure 5(a). The profitable bound for m_U is $[f^l(m_U), f^u(m_U)] = [-2, 4]$, since there are 2 objects below $\mathcal{BP}$ and 4 above it. This bound implies that there could be points on $\mathcal{BP}$ with profitability as low as -2 and as high as 4. Assume that we select object o_1 to refine m_U. Using o_1, m_U is split into three MBRs $\{m_1, m_2, m_3\}$ based on where the $(d-1)$-dimensional orthogonal hyperplanes, which pass through o_1, intersect $\mathcal{BP}$.Shrinking the end-points of the new MBRs to tightly enclose the part of $\mathcal{BP}$ they intersect is described in Appendix A. After the refinement process, from m_U's bounds, we can derive loose bounds for the three new MBRs as $[-2, 3]$, $[-1, 4]$, and $[-2, 3]$, respectively.[4] The upper bounds of m_1 and m_3 are derived by subtracting 1 from m_U's upper bound since m_1 and m_3 do not dominate the current object o_1; on the other hand, the lower bound of m_2 is increased by 1, since m_2 dominates o_1. We will remove m_U from our candidate list and only keep m_1, m_2 and m_3. Next if we select o_6 as the next object to be processed, m_2 is further split by o_6 into three MBRs $\{m_{2a}, m_{2b}, m_{2c}\}$ and their bounds become $[0, 4]$, $[-1, 3]$, and $[0, 4]$, respectively. The upper bound of m_{2b} is 3 because it is dominated by o_6. Besides generating these new MBRs, we also need to update the bounds of m_1 and m_3 to $[-1, 3]$ and $[-1, 3]$ since they are not dominated by o_6.

Note that a drawn hyperplane is not necessarily relevant to any MBR it intersects. Definition 8 formally shows whether a hyperplane is relevant to an existing MBR. Broadly speaking, a hyperplane drawn from object o is relevant to an MBR m if the profitability of m is affected by o; i.e., o dominates some $o' \in m$ or o is dominated by some $o' \in m$. For instance, in Figure 6, o's hyperplane perpendicular to the y-axis intersects m_1 and m_2. However, it is only relevant to m_1, since o is dominated by some $o' \in m_1$ but o and m_2 are incomparable.

Definition 8 (Relevant hyperplane). A hyperplane from an object o is relevant to an MBR m if and only if 1) the hyperplane intersects m and 2) there is at least one point o' in m, such that o' dominates o if $o \in \mathcal{O}^+$ or o' is dominated by o if $o \in \mathcal{O}^-$.

[4] Tight bounds cannot be derived unless we know the locations of the remaining objects. This algorithm refines MBRs by accessing the objects one-by-one; tight bounds for all finest MBRs will be established eventually after having accessed all objects.



(a) Refined by o_1 (b) Refined by o_6

Fig. 5. Example of MBR Refinement

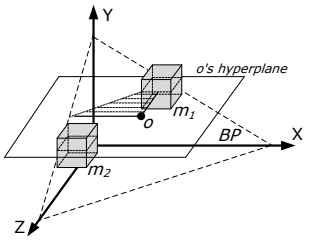


Fig. 6. Example of relevant hyperplane

Algorithm 1 MBR Refinement Algorithm

algorithm refine(MBR set $\mathcal{M}$, Budget Plane $\mathcal{BP}$, Object set $\mathcal{O}$)
1: **for all** object $o \in \mathcal{O}$ **do**
2: $\mathcal{M}$:=split($\mathcal{M}$, o, $\mathcal{BP}$) ▷ split the MBRs in $\mathcal{M}$ using o
3: **for all** MBR $m \in \mathcal{M}$ **do**
4: remove m and goto line 3 if $m.b^u < \gamma$
5: update $[m.b^l, m.b^u]$ based on o
6: set $\gamma := m.b^l$ if $\gamma < m.b^l$
7: $\mathcal{M}$:=$maximizeMBR(\mathcal{M})$

Lemma 2 (Pruning). An MBR m cannot contain a region which is part of the GenBOQ solution, if $m.b^u < \gamma$, where $\gamma = \max_{m \in \mathcal{M}} m.b^l$ and $\mathcal{M}$ is the set of all candidate regions.

Proof. Assume that there is a region r in m, which is part of the GenBOQ solution and $m.b^u < \gamma$. This means that r has higher or equal profitability than γ. Nevertheless, $r \in m$ and $f^u(r)$ must be smaller than or equal to $m.b^u$. It contradicts our assumption, so r does not exist. □

Algorithm 1 is a pseudocode of the MBR refinement algorithm. We start by initializing the result of GenBOQ as $\mathcal{M} = \{m_U\}$, where m_U is the MBR of the entire space. At every loop, we select an object o from $\mathcal{O}$ and split the current set of MBRs $\mathcal{M}$ according to the projected lines (orthogonal hyperplanes) from o (only relevant MBRs according

to Definition 8 are split). Then, we update the profitability bound $[m.b^l, m.b^u]$ for every MBR $m \in \mathcal{M}$ according to the location of o. During the process, we keep track of the highest lower bound γ among all MBRs in $\mathcal{M}$ and use it to prune MBRs m for which the upper bound is smaller than γ, based on Lemma 2.

Postprocessing. Recall that during MBR refinement, the newly created MBRs are shrunk as described in Appendix A. After Algorithm 1 terminates, the MBRs which are part of the solution are not essentially the largest possible. In a postprocessing step (Line 7 in Algorithm 1, $maximizeMBR(\mathcal{M})$), we enlarge the MBRs (and the feasible regions inside them) such that no point in space having the maximum profitability is missed in the result. The enlargement process is related to the relevance concept of Definition 8. Since the profitability of an MBR is only affected by relevant hyperplanes, the boundaries of the MBR which are not bounded by relevant hyperplanes are enlarged until the closest relevant hyperplane is met. The enlargement process does not affect correctness since profitability of enlarged MBRs does not change.

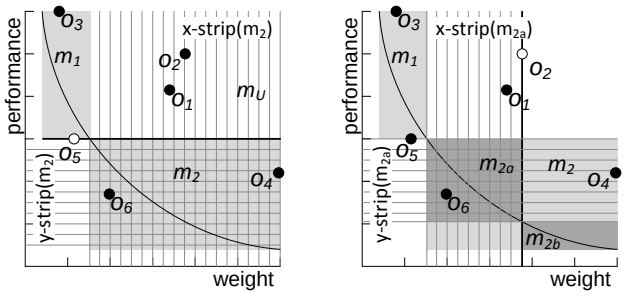
4.2. Divide-and-Conquer

The MBR refinement algorithm becomes expensive in high dimensional problems, due to its exponential space complexity (see Lemma 1). Our second approach is based on a divide-and-conquer (D&C) framework, which recursively decomposes the problem into smaller ones, until they become simple enough to be solved directly. Our approach is based on the observation that an object only affects the refinement of MBRs which intersect it in at least one dimension (e.g., in Figure 7(a), o_3 does not affect the refinement of m_2). Given an MBR m, Definition 9 defines the areas which influence its refinement as *strips*. Figure 7(a) shows two MBRs m_1 and m_2. The x-strip of m_2 is illustrated by the vertical lines that go through m_2 and the y-strip of m_2 is shown by the horizontal lines.

Definition 9 (Strip of an MBR). A k-strip of an MBR m is a hyper-rectangle, for which the lower and upper bounds in dimension k are derived from m; in other dimensions, the strip covers the entire space.

At every round, D&C takes as input an MBR m and finds the strip s of m containing the largest number of objects. Let k be the dimension of this strip, D&C takes the median object in the k-strip. If the median object is not relevant to m (see Definition 8), then next/previous objects to the median are accessed until a relevant object is found. Next, a hyperplane is drawn from the relevant object along dimension k which splits m into two MBRs. The effect is that the k-strips of the new MBRs will contain fewer objects compared to the k-strip of m. Starting from a single MBR m_U, Figure 7(a) shows how we can derive two new partitions, m_1 and m_2, by splitting the y-strip of m_U, using o_5 (the median in the y dimension). Figure 7(b) shows how m_2 is split into MBRs m_{2a} and m_{2b} based on o_2 (i.e., the median x-value in the x-strip of m_2).

Algorithm 2 includes the details of the D&C algorithm. Before running the algorithm, we sort all objects at each dimension and keep d sorted lists SL. Sorted list SL_k facilitates allocation of the objects in the k-strip of an MBR efficiently, by just defining split boundaries on SL_k. In addition, sorting helps to find the median points efficiently. Starting from $m = m_U$, in the first call, the algorithm first selects the median object o from the strip s_i of m that has the maximum number of objects. Next, function *planesplit* splits m using the orthogonal hyperplane that passes through o, defined by all points having the same value as o in dimension i. This is similar to the *split* function in Algorithm 1. The only difference is that function *planesplit* only projects one orthogonal



(a) m partitioned by o_5 (b) m_2 partitioned by o_2

Fig. 7. Example of Divide and Conquer

Algorithm 2 Divide and Conquer Algorithm

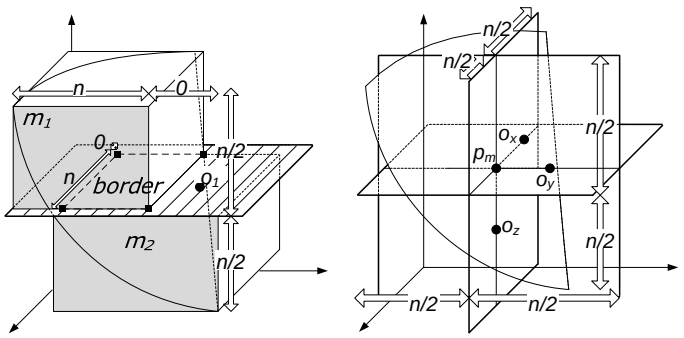
prerequisite
for every dimension i, objects are sorted and stored in sorted list SL_i
algorithm DC(partition m, Budget Plane $\mathcal{BP}$)
1: select a strip s_i of m such that s_i has the maximum number of objects
2: select the median object o from s_i using SL_i
3: $\mathcal{M}'$:=planesplit($\{m\}$, o, i, $\mathcal{BP}$)
4: **for all** $m' \in \mathcal{M}'$ **do**
5: update the bound of m' and prune m' if it violates Lemma 2
6: **if** the total number of objects in m''s strips $<$ threshold θ **then**
7: refine($\{m'\}$, $\mathcal{BP}$, {all relevant objects in the strips})
8: **else**
9: DC(m', $\mathcal{BP}$)

hyperplane instead of all hyperplanes as in function *split*. For instance, o_5 projects one line to $\mathcal{BP}$ parallel to the x-axis but no line parallel to the y-axis in Figure 7(a).

For each of the two new MBRs m' created by *planesplit*, we update the tight profitability bound of m' by scanning all objects in the strips of m'. We decide to switch to the MBR refinement algorithm on a partition m', if the total number of objects in all strips of m' is smaller than a given threshold θ. Otherwise, m' is further split by calling Algorithm 2 recursively. In this case, the refinement algorithm only accesses objects within the strips of m' in order to refine it to finest MBRs. Therefore, the space complexity of calling this method for m' is only bounded by the number of finest MBRs in m'.

4.3. Optimizing Divide-and-Conquer

During D&C, a d-dimensional MBR is split into at most two MBRs by a $(d-1)$ dimensional orthogonal hyperplane. At each recursion, there are at most two calls. Thus, D&C is a depth-first search algorithm with low space complexity. On the other hand, the resulting MBRs only have disjoint strips in one dimension in the worst case. Consider the 3D example of Figure 8(a). Two MBRs, m_1 and m_2, are created by the vertical hyperplane from the median object o_1 of m_U in the y dimension. Note that m_1 and m_2



(a) use 1 median hyperplane (b) use d median hyperplanes

Fig. 8. Analysis of Divide and Conquer partitioning

share a *border* area (instead of a single point in the 2D example of Figure 7(a)) in the x and z dimensions. Therefore, after the split, in the worst case, all objects in all dimensions but y may fall into both MBRs' strips. Assuming that we have n objects in m_U, the total number objects in m_1's (m_2's) strips is up to $n/2 + (d-1)n$=$(d-0.5)n$. This means that, in the worst case, the input fed to the D&C subproblems may be reduced very slowly (from dn to $(d-0.5)n$ at each recurrence).

Avoiding the Worst Case. In order to avoid the worst case discussed above, we can adapt D&C to split the current MBR m into 2^d new MBRs, such that for every dimension k, if the k-strip of m has n objects, the k-strip of each new MBR m' will have $n/2$ objects. This is possible if, for every dimension k, we select the median object in the k-strip of m and form a point p_m having as coordinates these median values. The space is then split by drawing k hyperplanes perpendicular to each dimension k, passing through p_m. Figure 8(b) shows a 3D example, where the three median objects o_x, o_y, and o_z all dimensions are used to form p_m, which then splits the MBR into 8 new ones, each having $n/2$ objects in each of its strips.

Optimization I - Optimizing Split Selection. Although the above 2^d-partitioning strategy avoids the worst case, it is a pessimistic approach that will not take advantage of potential best cases. Note that the best case of the simple D&C algorithm creates only two recurrence calls, each having only $O(d \cdot n/2)$ input size (i.e., all objects are partitioned equally at each dimension by *one planesplit*). As we show in the Section 4.5, the time complexity of such a best case is only $O(d \cdot n \log(d \cdot n))$. Therefore, if a *planesplit* partitions a MBR into two MBRs, such that their k-strips in all dimensions have small overlap, we perform this binary split at the D&C call; otherwise, we perform a 2^d-split using p_m. In the example of Figure 8(a), to assess the quality of the y-*planespit* based on o_1, we compute the number of objects co-existing in the x-strips of m_1 and m_2 and sum it to the corresponding overlap in the z-strip. Dividing this number by the sum of objects in the x- and z-strips of the original MBR m, gives us an *overlap ratio* for the y-*planespit*. If this ratio is smaller than a parameter ρ, then we choose to perform the y-*planesplit* over the 2^d-split.

This version of D&C (shown as Algorithm 3) avoids the worst case, while selectively choosing binary splits if they are favorable. For each dimension k, we find the median point of k-strip, update p_m and keep track in k_{best} the dimension for which the binary split results in the minimum overlap ratio. Computing the overlap ratio is done very

Algorithm 3 Optimized Divide and Conquer Algorithm

algorithm DC(partition m, Budget Plane $\mathcal{BP}$)
1: **for** every dimension k **do**
2: select the median object o_k from k-strip of m and update p_m
3: $\mathcal{M}'$:=planesplit($\{m\}$, o_k, k, $\mathcal{BP}$)
4: **if** overlap ratio of split is better than previous splits **then**
5: o_{best}:=o_k; k_{best}:=k
6: **if** overlap ratio of $o_{best} < \rho$ **then**
7: $\mathcal{M}'$:=planesplit($\{m\}$, o_{best}, k_{best}, $\mathcal{BP}$)
8: **else**
9: $\mathcal{M}'$:=split($\{m\}$, p_m, $\mathcal{BP}$)
10: Run lines 4 to 9 in Algorithm 2

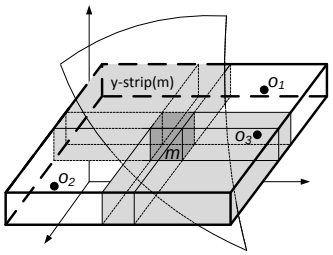


Fig. 9. Example of relevant objects in a strip

efficiently by performing binary search on the corresponding sorted lists SL to identify the index of the first and last object at each dimension for the strips of m_1 and m_2. If the best overlap ratio is smaller than a given threshold ρ, we perform the corresponding *planesplit*, otherwise we do a 2^d *split* based on p_m.

Optimization II - Removing Irrelevant Objects. Note that there could be some irrelevant objects inside the strips of an MBR. For instance, there are 3 objects inside the y-strip of MBR m in Figure 9. Objects o_1 and o_2 are irrelevant to m since the profitability of m is not affected by them (see Definition 8). Thus, only objects in the shaded area of y-strip(m) (e.g., o_3) are considered in the split process of m.

In the second optimization of D&C, instead of being modeled by the boundaries of a hyper-rectangle, each strip of an MBR m keeps an explicit set of object IDs if these objects are relevant to m and they are inside the strip. For each D&C call of an MBR m (at Line 9 in Algorithm 2), m collects all relevant objects from its parent for each strip. With this modification, all irrelevant objects are removed from subsequent refinements of m and the performance improves significantly. Each new D&C call keeps at most $(d - 0.5)n$ IDs and during the algorithm we may store up to $O(dnh)$ IDs, where h is the depth level of the D&C process. Thus, the space complexity becomes $O(\max\{dnh, \theta^{d-1}\})$.

Additional Optimizations. Two more optimizations are included in our D&C implementation. First, we observe that Lemma 2 is more effective if a better γ is found early. Based on this observation, at each D&C call, the generated MBRs are processed in descending order of their lower bounds $m_i.b^l$. Second, if there are K objects in the datasets having the same feature values, then these K objects can be grouped into one

artificial object with *capacity* K. The lower and upper bounds are updated according to the capacities of objects throughout the D&C computation. This technique is effective for real data with low-cardinality domains, as shown in the experimental section.

4.4. Hybrid Approach

D&C reduces the space complexity by dividing the search space recursively, in a depth-first search fashion. In this section, we explore the application of a best-first refinement (BFR) algorithm and a hybrid solution that combines BFR and D&C.

BFR is motivated by the observation that it is not necessary to refine the MBRs in a depth-first fashion. After refining a MBR m using one *planesplit* of the D&C method, the upper bounds of the resulting MBRs may be significantly smaller than that of m. In this case, it might be more promising to switch to the refinement of another MBR from those computed so far, which has higher upper bound than the new MBRs. BFR follows such a best-first strategy for refining the MBRs. First, BFR pushes MBR m_U that covers the entire space into a heap $\mathcal{H}$, which organizes the MBRs in descending order to their upper bounds. At each iteration, the MBR with the highest upper bound is deheaped, we split it using a D&C hyperplane, and the resulting MBRs are pushed back on the heap if they are not finest. The profitability of the best finest MBR found is used as a termination bound γ; the algorithm terminates when the next deheaped MBR has an upper bound which is lower than γ.

Algorithm 4 Best First Refinement Algorithm

algorithm BFR(partition m, Budget Plane $\mathcal{BP}$)
1: push an MBR m_U that covers the entire space into a heap $\mathcal{H}$
2: **while** $\mathcal{H}$ is not empty **do**
3: $m := \mathcal{H}.pop()$
4: run lines 1 to 9 in Algorithm 3
5: push all MBRs in $\mathcal{M}$ to $\mathcal{H}$
6: break the loop if the top MBR in $\mathcal{H}$ violates Lemma 2

The pseudo code of BFR is shown in Algorithm 4. BFR always chooses the currently best MBR to refine. However, this approach becomes very expensive or infeasible if $\mathcal{H}$ grows beyond the memory limits (recall that the number of possible MBRs could be as high as $O(d \cdot n^{d-1})$).

To alleviate the high memory consumption of BFR, we propose a hybrid method that combines the advantages of BFR and D&C. A variable ω is used to control the memory usage of BFR. BFR is initiated and used while $|\mathcal{H}| \leq \omega$. As soon as $|\mathcal{H}| \geq \omega$, the MBRs are accessed according to their order in $\mathcal{H}$, and D&C is executed for each of them to derive the finite MBRs included in them with their profitabilities. We update γ and continue executing D&C for next MBR in $\mathcal{H}$ as long as the next MBR in $\mathcal{H}$ has an upper bound not smaller than γ. As we show experimentally, with a careful selection of parameter (ω), this hybrid solution can outperform D&C.

4.5. Complexity analysis

MBR Refinement Algorithm. At each step, the algorithm reads an object o and refines one or more MBRs (e.g., m_2 in Figure 5(b)). In addition, it updates the profitability

bounds of one or more MBRs (e.g., m_1 and m_3). Therefore, at each step we may have to update or refine all existing MBRs. In the worst case, the algorithm terminates after generating all $O(d \cdot n^{d-1})$ finest MBRs, for a problem with n d-dimensional objects. The overall worst-case complexity is $O(n \cdot d \cdot n^{d-1}) = O(d \cdot n^d)$. The space complexity of the algorithm is $O(d \cdot n^{d-1})$, as all MBRs have to be maintained and compared with the current point at each step.

Divide-and-Conquer. First, we analyze the computational complexity of the 2^d partitioning heuristic of D&C, which as explained has the best worst-case cost. Suppose that we have n objects, finding the median object o from each strip takes $O(\log n)$ time, since the objects are stored in the strip in sorted (passed from the previous D&C call). The most expensive step is to scan all strips of the divided MBR m and update the bound of each new MBR. Thus we scan $O(d \cdot n)$ objects in total in the d strips, and each object is used to update the bounds of the new $2^d - 1$ partitions (note that one of the 2^d partitions is guaranteed not to intersect $\mathcal{BP}$, therefore it is pruned). Thus, the total time of a D&C call is $O((2^d - 1) \cdot d \cdot n)$ and each recursive call has $O(d \cdot n/2)$ input size. Note that the input size of a D&C call is $d \cdot n$ instead of n since an MBR has d strips in total. By setting $N = d \cdot n$, the cost of D&C can be described by the following recurrence.

$$T(N) = (2^d - 1)T(N/2) + O((2^d - 1)N) \tag{4}$$

According to the Master Theorem (Cormen, Leiserson, Rivest and Stein, 2009), the bound of the recurrence function is $\Theta(N^{\log(2^d-1)}) = \Theta((d \cdot n)^{\log(2^d-1)})$. In addition, the space complexity is reduced from $O(d \cdot n^{d-1})$ to $O(\max\{2^d \log d \cdot n, |\mathcal{R}|, \theta^{d-1}\})$, where $\log d \cdot n$ is the depth of the execution stack keeping 2^d MBRs at each level, $\mathcal{R}$ is the maximum size of the candidate result set of the query, and θ is the bound used to switch to the MBR refinement process. In summary, the worst-case time complexity is higher than that of the MBR refinement algorithm. However, this algorithm is more efficient in practice, due to its limited space requirements.

We now analyze the best case of the algorithm. This happens when, at every call, the input MBR m is partitioned by a *planesplit* equally into two MBRs, each having the input size of m divided by 2. Therefore, the cost of D&C in the best case can be described by the following recurrence.

$$T(N) = 2T(N/2) + O(2N) \tag{5}$$

By applying Master Theorem (Cormen et al., 2009), the bound of the recurrence is $\Theta(N \log N) = \Theta((d \cdot n) \log(d \cdot n))$. As discussed in Section 4.3, the optimized D&C method would prefer good binary splits to 2^d splits if they result in good partitions.

5. Approximation of GenBOQ

Based on our D&C framework, we propose a method which returns an approximate GenBOQ result with a quality guarantee. Instead of computing all feasible regions in GenBOQ, our approximation solution (ApprGenBOQ) returns only one feasible region r_{appr} such that the profitability of r_{appr} is bounded by a tolerance T from the optimal. In other words, the MBR returned by the approximate method does not have to be finest due to the tolerance bound T. Thus, an MBR is a potential result only if the gap between its upper and lower profitability bounds is smaller than T. Formally:

Definition 10 (Qualified MBR). An MBR m is a potential result of ApprGenBOQ if $m.b^u - m.b^l \leq T$.

In the D&C framework, we iteratively split an MBR if the number of objects in the MBR's strips is larger than a threshold θ. Otherwise, we refine the MBR by Algorithm 1. For the approximation solution, if an MBR is qualified, it is not necessary to split/refine it further. However, a qualified MBR is not necessary a result of ApprGenBOQ. We use Definition 11 to compare two qualified MBRs and eventually find a qualified MBR that is a result.

Definition 11 (Comparison of qualified MBRs). We say qualified MBR m_1 is better than qualified MBR m_2 if and only if $m_1.b^u > m_2.b^u$.

In Lemma 3, we show that the best among all qualified MBRs (according to Definition 11) must be a result of ApprGenBOQ. Note that we return only one qualified MBR in ApprGenBOQ. Therefore, if the upper bound of a new created MBR m' is not larger than the maximum upper bound η of the qualified regions found so far, then m' can be pruned safely. In summary, once we compute a qualified MBR, the MBR is either kept as a result candidate or pruned by η.

Lemma 3 (Correctness of approximation). A qualified MBR m is a result of ApprGenBOQ if $m.b^u \geq \eta$, where $\eta = \max_{m \in \mathcal{M}_{qual}} m.b^u$ and $\mathcal{M}_{qual}$ is the set of qualified MBRs.

Proof. Suppose that the maximum profitability is p. So our task is to prove that $p \in [m.b^l, m.b^u]$.

- If $p < m.b^l$, then there must exist a profitability $p' = f(x)$ better than p, where $x \in m$. This contradicts the assumption that p is the maximum profitability.
- If $p > m.b^u$, then we must have another MBR m' where $p \in [m'.b^l, m'.b^u]$, which means $m.b^u < p \leq m'.b^u$, so m should be pruned according to Definition 11. This contradicts the fact that m is the last result candidate.

□

According to the problem definition, we return only one qualified MBR in ApprGenBOQ. Therefore, if the upper bound (as well as the lower bound) of new created MBR m' is worse than the maximum upper bound of all qualified regions seen so far, then m' can be pruned safely since m' is not better than the current candidate according to Definition 11. In summary, once we compute a qualified MBR, the MBR is either kept as a result candidate or pruned by the lemma.

Algorithm 5 (extended from Algorithm 3) is a pseudocode of our approximation solution. For each newly created MBR m', we first test whether m' can be pruned or not (Line 4). If m' is not pruned and m' is a qualified MBR, then m' is set as the result candidate. If m' is not a qualified MBR, m' is either refined or split by corresponding processes. Note that, during the MBR refinement process, we stop refining MBRs once they become qualified. Accordingly, function *refineAppr* in Line 8 is used to represent the modified version of Algorithm 1. After the refinement process, we may update the result candidate from the newly qualified MBRs (Line 9).

Note that our approximation solution is directly applicable to the hybrid approach. The extension is trivial so that we omit the details for the conciseness of the manuscript.

Algorithm 5 Divide and Conquer Approximation Algorithm

algorithm DCAppr(partition m, Budget Plane $\mathcal{BP}$, tolerance T)
1: Run lines 1 to 9 in Algorithm 3
2: **for all** $m' \in \mathcal{M}'$ **do**
3: update the bound of m'
4: prune m' if it violates Lemma 2 or $m'.b^u < \eta$
5: **if** $m'.b^u - m'.b^l \leq T$ **then**
6: set m' as the result candidate
7: **else if** the total number of objects in m''s strips $<$ threshold θ **then**
8: refineAppr($\{m'\}$, $\mathcal{BP}$, {all relevant objects in the strips})
9: update result candidate
10: **else**
11: DCAppr(m', $\mathcal{BP}$, T)
12: return result candidate

6. Experimental Evaluation

We empirically evaluate the performance of the proposed algorithms: the MBR refinement algorithm (MBR) (Section 4.1), divide-and-conquer (D&C) (Section 4.2), the hybrid approach (Section 4.4), and the approximation method (Section 5).

Three types of synthetic datasets, *independent* (IND), *correlated* (COR), and *anti-correlated* (ANT), are generated according to the methodology in (Börzsönyi et al., 2001). In IND datasets, the feature values are generated uniformly and independently. COR datasets contain objects whose values are correlated in all dimensions. ANT datasets contain objects whose values are good in one dimension and tend to be poor in other dimensions. In addition, we generate *cluster* (CLU) datasets by randomly selecting ξ independent objects, and treat them as cluster centers. Each cluster object is generated by a Gaussian distribution with mean at the selected cluster center and standard deviation 5% of each dimension domain range. We set ξ to 10 by default. The above four types of data are common benchmarks for skyline queries (Börzsönyi et al., 2001; Zhang, Mamoulis and Cheung, 2009). Our dataspace contains d dimensions (in the range from 2 to 5). Additionally, we experiment with two real datasets, Household (*Household dataset*, 2008) and NBA (*NBA Basketball Statistics*, 2009).

By default, we use one type of constraint function, $C(x) = \frac{1}{\Pi_{1 \leq i \leq d}(x[i])}$, which gives equal weights to all dimensions, in order not to bias any attribute. In general, the constraint function is designed by a domain expert. In addition, we also evaluate an alternative function, composed by a set of linear functions.

All methods were implemented in C++ and the experiments were performed on an Intel Core2Duo 2.66GHz CPU machine with 4 GBytes memory, running on Ubuntu 9.04. Table 3 shows the ranges of the investigated parameters, and their default values (in bold). In each experiment, we vary a single parameter, while setting the others to their default values. For each method, we measure the total execution time, including any preprocessing costs, and the peak memory usage. As we will show shortly, after tuning of parameters θ (refinement threshold) and ρ (overlap ratio), they are set to their best values (10 and 0.7, respectively), throughout all remaining experiments.

Parameter Tuning. We first study the effect of the various tuning parameters on the algorithms. We investigate the effect of θ (Section 4.2) and the overlap ratio ρ (Section 4.3). Figure 10(a) shows the effect of θ on the cost of the D&C algorithm on the default IND dataset. For very small values of θ, the refinement process becomes very fast;

Table 3. Range of parameter values

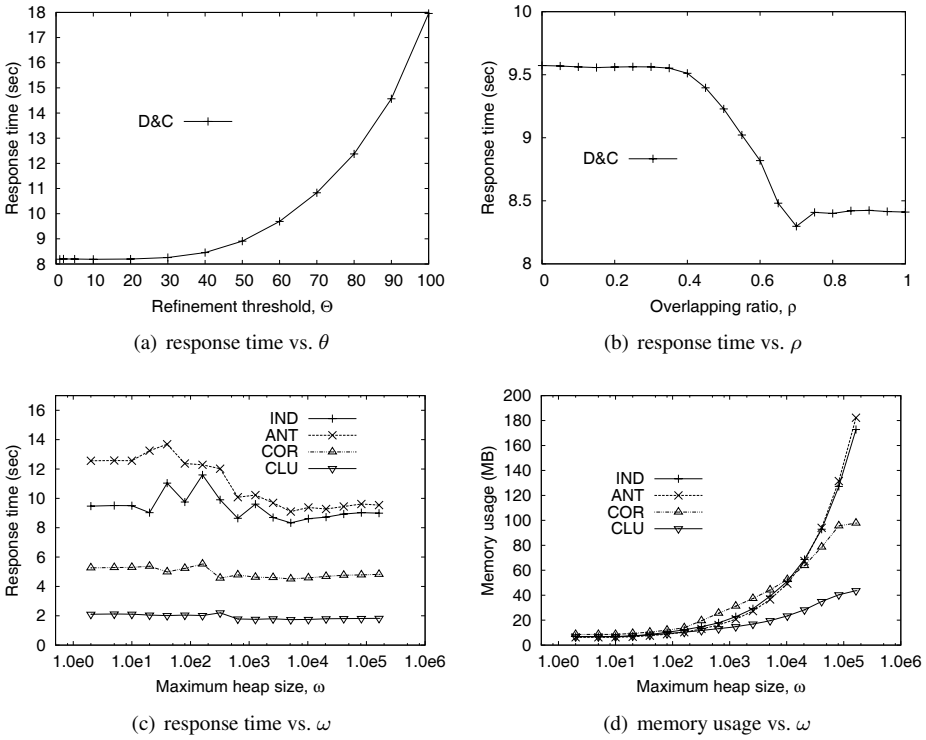
Parameter	**Values**
$\|\mathcal{O}\|$ (in thousand)	10, 25, **50**, 100, 200
Dimensionality d	2, 3, **4**, 5, 6
Data distribution	**IND**, ANT, COR, CLU
Overlap ratio ρ	0, 0.05, 0.1, ..., **0.7**, ..., 0.95, 1
Refinement threshold θ	2, 5, **10**, 20, 40, 80
Maximum heap size ω	0, 2, 5, 10, 20, ..., **5120**, ..., 81920, **163840**
Number of clusters ξ	**10**, 50, 100, 200,..., 1600
Constraint function $C(o)$	**(a)** $\frac{1}{\Pi_{1\leq i\leq d}(o[i])}$ (b) $\max\left\{\begin{array}{c}\frac{1}{\alpha\cdot o[1]+\sum_{1\leq i\leq d, i\neq 1} o[i]}\\ \cdots \\ \frac{1}{\alpha\cdot o[d]+\sum_{1\leq i\leq d, i\neq d} o[i]}\end{array}\right.$
Development budget B	2, 5, 10, **100**, 1000, 10000, 100000
α	0.1, 0.2, 0.4, 0.8, 1.6, 3.2
Quality $T/\|\mathcal{O}\|$	0.01%, 0.05%, 0.1%, 0.5%, **1%**, 2%, 5%
Discrete values (DADA)	5, 7, **10**, 12, 15, 17, 20

however, there are also more D&C recurrence calls. For very large θ, the execution time is high due to the expensive refinement process. Also, the memory usage becomes higher; the peak number of MBRs created by the refinement process at $\theta = 100$ is 21541 while it is only 35 and 169 at $\theta = 10$ and $\theta = 20$ respectively. The best overall value in terms of response time is in between 5 to 20, so we choose $\theta = 10$ as the default value in order to reduce the number of D&C recurrence calls.

Next, we test the effect of the overlap ratio ρ on the D&C algorithm. As Figure 10(b) shows, the cost trend of D&C agrees with our discussion in Section 4.3. D&C becomes more expensive when we only use 2^d splits ($\rho = 0$) since it does not take advantage of possible best cases. Its execution time is 15% higher than our default ratio $\rho = 0.7$.

We also study the effect of the maximum heap size ω on the hybrid solution. Figures 10(c) and 10(d) show the response time and memory usage of different data distributions as a function of ω. For $\omega = 0$, hybrid becomes plain D&C; on the other hand, as ω increases, hybrid becomes more like BFR. COR and CLU datasets favor the BFR approach since the data are more skewed and a good γ can be computed early. For IND and ANT datasets it is more difficult to find a good ω value, since the points on the budget plane have similar profitabilities and a good bound to prune large MBRs is hard to find. For all data distributions, the memory usage increases with ω. We set ω to 5120 since it gives the minimum value for all data distributions, regardless of the large memory consumption compared to plain D&C method.

Effectiveness of Optimizations. Next, we evaluate the effectiveness of the optimizations proposed in Section 4.3 within the D&C algorithm. The fully optimized algorithm (D&C) is compared against D&C-NoWorst (the divide-and-conquer algorithm with all 2^d splits, described in Section 4.3 *Avoiding the Worst Case*) and D&C-OptSplit (the basic divide-and-conquer algorithm with optimized split selection, described in Section 4.3 *Optimizing Split Selection*).



(a) response time vs. θ (b) response time vs. ρ

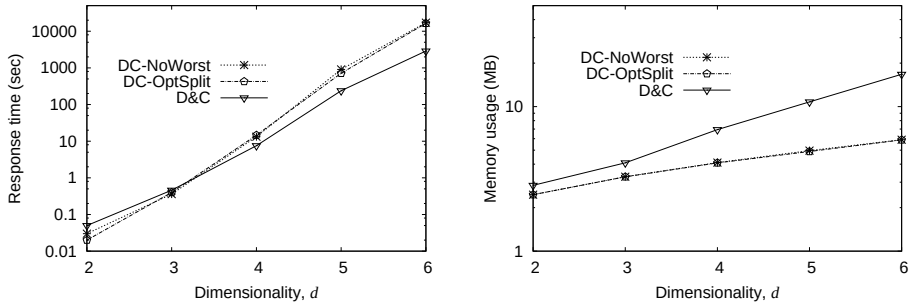
(c) response time vs. ω (d) memory usage vs. ω

Fig. 10. Sensitivity experiments, IND

Figure 11 plots the response time and memory usage of the three D&C versions as a function of the dimensionality d, after setting all other parameters to their default values. The response time increases exponentially in all methods, but fully optimized D&C is less sensitive to dimensionality than the other methods. It is 5.94 and 5.7 times faster than D&C-NoWorst and D&C-OptSplit for $d = 6$, while consuming at most 1.85 times more memory. Note that D&C-NoWorst and D&C-OptSplit are faster than the D&C at $d = 2$, since the computation in this case is very fast and the optimization for irrelevant objects pruning comes with a significant cost factor.

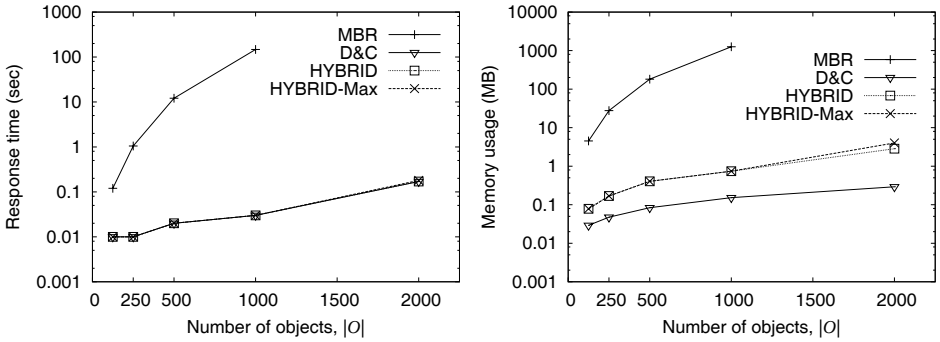
Scalability experiments. Next, we compare the proposed solutions (D&C and hybrid) to the naive method (MBR). Two versions of the hybrid method are evaluated. In HYBRID, ω is set to 5120 based on our tuning. HYBRID-Max is used to test BFR when ω is set to a large number ($\omega = 163,840$).

Because the memory usage of the baseline method MBR grows exponentially with the dimensionality, we only test it on a small dataset with $|\mathcal{O}|$ ranging from 125 to 2000, for $d = 4$. As shown in Figure 12, the cost of MBR grows very fast. When $|O| = 2000$, the space requirements of MBR exceed the available memory. The cost of the other methods grows exponentially, as well, but at a much lower pace than the naive approach. Their memory usage increases at even lower pace. This demonstrates that our approaches are a substantial contribution, since this is a very expensive problem even for small datasets.



(a) response time (b) memory usage

Fig. 11. Varying dimensionality d, IND

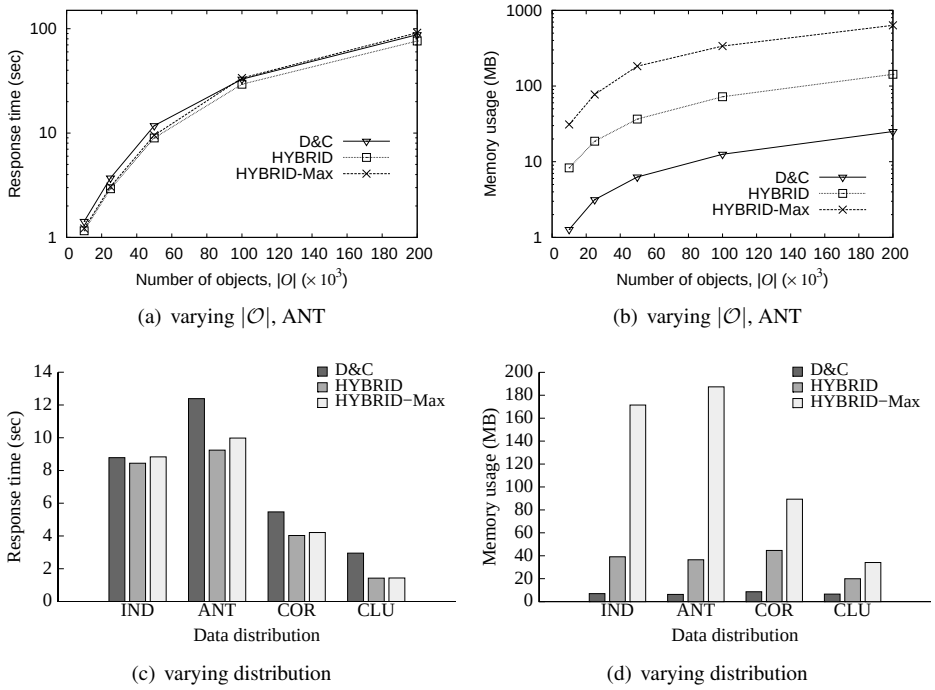


(a) response time (b) memory usage

Fig. 12. varying $|\mathcal{O}|$ for small cases, IND

In the remaining experiments, we exclude the baseline method (MBR), due to its high cost. Figure 13(a) shows the response time of the remaining methods as a function of the number of objects on anti-correlated data. HYBRID is 14.9% (19.9%) faster than D&C (HYBRID-Max) at $|\mathcal{O}| = 200$K. However, HYBRID consumes more memory than D&C (at least 5.7 times) due to the heap structure. More precisely, it consumes 72.24 MB (142.75 MB) for $|\mathcal{O}| = 100$K ($|\mathcal{O}| = 200$K), while D&C consumes only 12.54 MB (25.05 MB). HYBRID-Max is not only slower but also consumes more memory (633.64 MB at $|\mathcal{O}| = 200$K) than the others. The memory usage of all methods increases linearly with $|\mathcal{O}|$, as shown in Figure 13(b).

Figures 13(c) and 13(d) show the response time and memory usage of the methods on different data distributions. HYBRID is the best method in terms of response time in all four data distributions IND, ANT, COR, and CLU. D&C is nearly as good as (only 4% slower) HYBRID on IND data while consuming 5.63 times less memory. Note that all methods perform better on skewed data, since a good γ can be found early. In summary, HYBRID provides the best query response time to GenBOQ in all datasets; however, it consumes more memory than D&C and the optimal ω value is difficult to set for different data distributions. Therefore, we recommend the use of D&C, since it does not depend on hard-to-set parameters, it has low memory consumption, and it is



(a) varying $|\mathcal{O}|$, ANT

(b) varying $|\mathcal{O}|$, ANT

(c) varying distribution

(d) varying distribution

Fig. 13. Scalability experiments

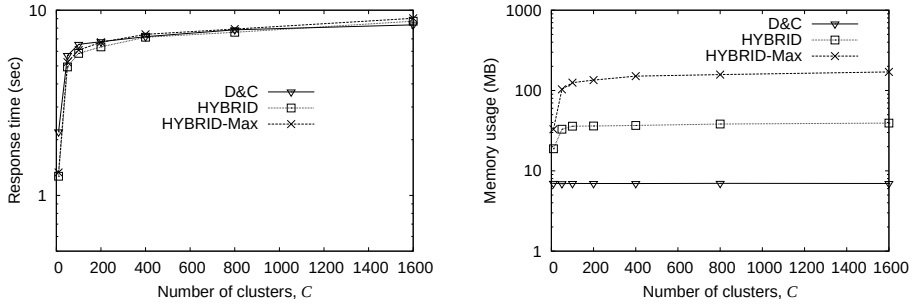
not a lot worse than the best tuned hybrid algorithm. In our evaluation, all methods have similar performances which the response time generally increases with the number of clusters ξ.

Data distribution. We also evaluated our methods for different numbers of clusters C on CLU data. Figure 14 demonstrates the response time and memory usage of all methods as a function of C. All three methods have similar performances which the response time generally increases with the number of clusters. HYBRID is the best method except at $C = 1600$ since the ω value of HYBRID is no longer optimized. The memory usage of all methods becomes stable when $\xi \geq 200$ since the data become uniformly distributed. The memory usage of D&C is not sensitive to ξ since the most memory consuming procedure, MBR refinement, is well controlled by the system parameter θ.

Constraint functions C. We study the performance of the methods for different constraint functions C. The constraint function can be either a single monotonic function or a set of monotonic functions.

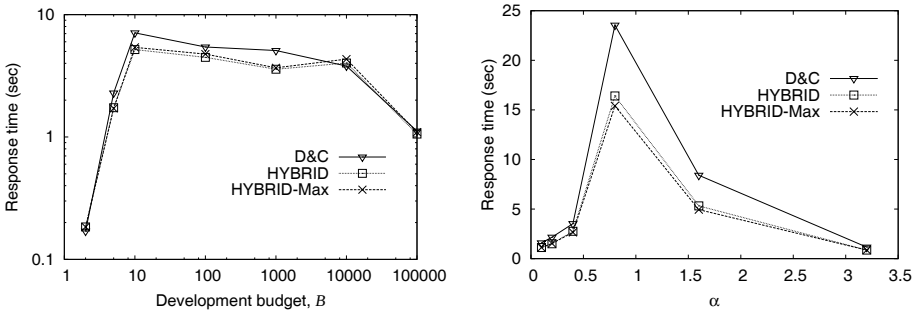
Figure 15(a) plots the response time of the solutions with respect to the budget B of a type-a function ($C(o) = \frac{1}{\Pi_{1 \leq i \leq d}(o[i])} = B$) on correlated datasets (COR). For very small or very large values of B, the problem becomes easier on COR datasets since the budget plane $\mathcal{BP}$ is close to left-bottom or right-top corner of the space. Hybrid methods outperform D&C on COR datasets.

We also evaluated the performance of type-b constraint functions (a set of linear functions) defined in Table 3 on IND datasets. For each linear function lf_i, we set all



(a) response time (b) memory usage

Fig. 14. Varying on number of clusters, ξ



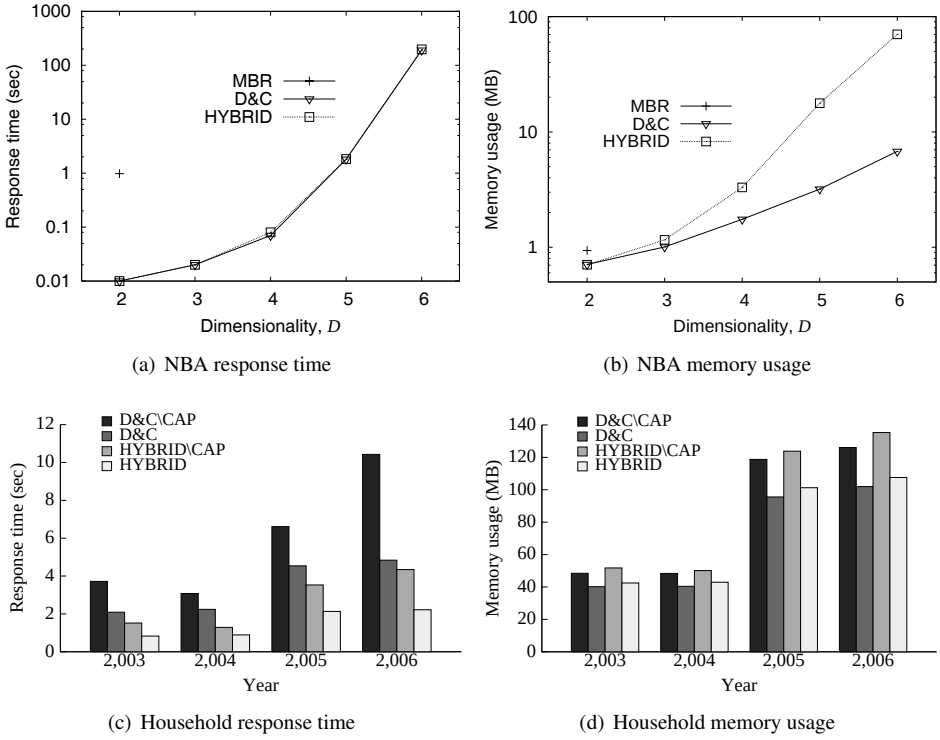
(a) Varying B on type-a, COR (b) varying α on type-b, IND

Fig. 15. varying on different constraint function types C

coefficients to 1 except i-th coefficient; the i-th coefficient is set to α. Besides, we set $B = 1$ in order to avoid negative profitability results such that the size of positive object set $\mathcal{O}^+$ is larger than negative object set $\mathcal{O}^-$ on these two datasets. As Figure 15(b) shows, the problem is easier to solve at $\alpha = 0.1$ and $\alpha = 3.2$, for the same reason as in small and large values of B in type-a constraint functions. Like in type-a functions, hybrid methods outperform D&C on independent datasets.

Real data. NBA contains 12,278 statistics from regular seasons during 1973-2008, each of which corresponds to the statistics of an NBA player's performance in 6 aspects (minutes played, points, rebounds, assists, steals, and blocks). Household consists of 3.6M records during 2003-2006, each representing the percentage of an American family's annual expenses on 4 types of expenditures (electricity, water, gas, and property insurance). In the following experiments, we exclude HYBRID-Max due to its bad performance in terms of response time and memory usage. To avoid negative profitability results, we set the development budget B to 1M.

Figure 16(a) shows the response time of three methods (MBR, D&C, and HYBRID) as a function of dimensionality. The response time of all methods increases exponentially to the dimensionality. D&C and HYBRID have similar response time due to small data size. Again, MBR exceeds the available memory when $d \geq 3$, so it is only included



(a) NBA response time

(b) NBA memory usage

(c) Household response time

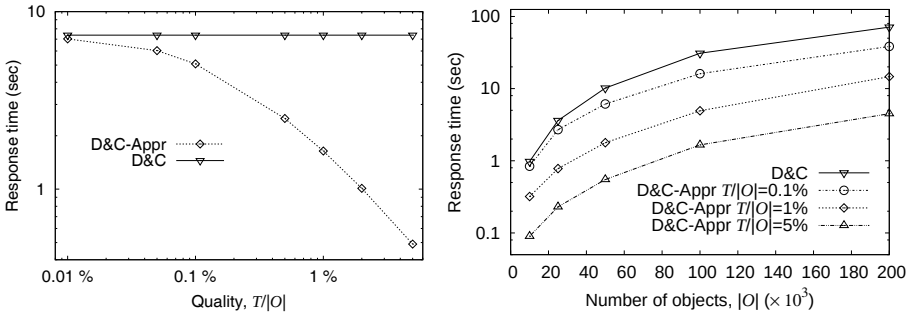
(d) Household memory usage

Fig. 16. Results with Real Datasets

for $d = 2$; in this case MBR is 2 orders of magnitude slower than our proposed methods. The memory consumption for NBA data is illustrated in Figure 16(b). From the figure we can see that, even for the only available $d = 2$ case, MBR consumes more memory than D&C and HYBRID. As dimensionality d increases, the memory consumption of D&C and HYBRID increases exponentionally, where D&C is consistenly better than HYBRID, consuming less memory than HYBRID with a ratio up to one magintude when $d = 6$.

Figure 16(c) compares the response time of D&C and HYBRID with their variants D&C\CAP and HYBRID\CAP that exclude the capacity optimization described in Section 4.3. We divided Household into four datasets with 516K, 514K, 1.25M, and 1.35M records from years 2003, 2004, 2005, and 2006 respectively. The feature values in Household are discrete, so there are some tuples having the same feature values in all dimensions; in this case the objects are grouped to a single capacitated object. The number of discrete objects are 242K, 250K, 520K, and 542K, respectively in the four years. The fully optimized methods are significantly faster than the methods that do not apply capacity optimization. HYBRID is faster than D&C in all tests since the Household dataset is highly clustered; this is also the reason why all methods perform better on the Household dataset than the synthetic datasets. Figure 16(d) demonstrates the effectiveness of our capacity optimization, which saves the memory usage by at least 14.4% as compared to the version without this optimization.

Approximation. We use the ratio of tolerance bound T divided by the number of



(a) varying quality $T/|\mathcal{O}|$, IND (b) varying $|\mathcal{O}|$, IND

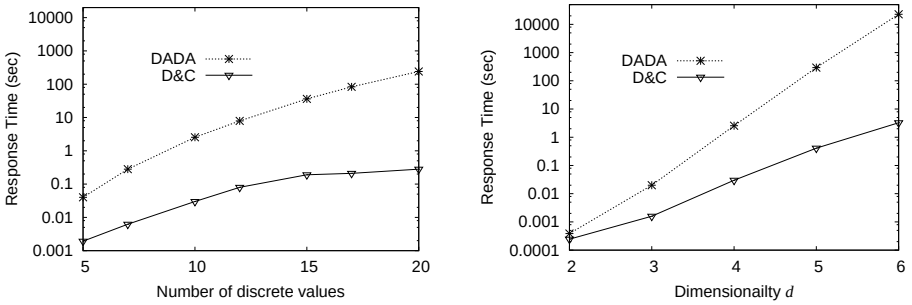
Fig. 17. Approximation Performance

objects $|\mathcal{O}|$ to represent the quality of an approximation result. We first evaluate the response time of our approximation approach as a function of quality, showing the response time of D&C as a baseline. Figure 17(a) shows that the response time of D&C-Appr decreases dramatically with the increase of T. At the default quality setting ($T/|\mathcal{O}| = 1\%$), D&C-Appr is 4.5 times faster than D&C, showing that D&C-Appr finds a result efficiently without sacrificing much precision. Figure 17(b) tests the scalability of D&C-Appr, showing the response time of four methods (D&C and three quality settings of D&C-Appr) as a function of the number of objects. All D&C-Appr methods are consistently faster than D&C. Moreover, D&C-Appr with $T/|\mathcal{O}| = 1\%$ is 4.86 times faster than D&C for $|\mathcal{O}| = 200$K. Thus, the performance gain of D&C-Appr is not affected by the problem size.

DADA. Finally, we test the efficiency of our approach against DADA (Li et al., 2006; Zhu et al., 2012). We generate datasets, where the feature values of objects are discrete for every dimension, so that both LOQ in DADA and GenBOQ return the same result. For each dataset, we construct the D*-tree index for LOQ. In Figure 18(a) and Figure 18(b), we show the response time of LOQ (excluding the time to construct the D*-tree) and our D&C approach. D&C outperforms DADA by 1 to 3 orders of magnitudes, and this superiority increases when we increase the number of discrete values or the dimensionality. This demonstrates why our method is more general than DADA, since it can be applied to continuous or discrete feature spaces of large cardinalities.

7. Conclusion

In this paper we studied a new optimization problem; given a set of existing products in the market, modeled by their feature values, we seek for the features to give to a new product such that its profitability is maximized. Profitability is modeled by the number of products which dominate and are dominated by the new product. We constrain the search space by a budget, which is monotone to the feature values. The solution of the problem then becomes a set of continuous regions in the feature space, which can be accurately expressed by minimum bounding rectangles (MBRs). We proposed and compared three methods for this problem. The first is a baseline MBR refinement approach, which iteratively processes the existing products and refines regions in space enclosing feasible feature vectors with the same profitability bounds. This method has



(a) varying discrete values (b) varying d

Fig. 18. Comparison with DADA

time and space complexity exponential to dimensionality. The second method applies depth-first search, refining regions in a divide-and-conquer (D&C) manner and has low space requirements. The third method is a hybrid method that applies best-first search in the D&C framework. Our experiments show that the last two methods scale much better than the baseline approach in practice, while having low memory requirements. Given the inherently high cost of the problem, we proposed an approximation method based on the D&C strategy. The experiments show that this method can find a result of 1% or less relative difference to the optimal much faster than exact approach. In the future, we plan to study the parallelism of GenBOQ to further reduce the computational cost; the D&C framework supports such a design. In addition, we will consider an extension of GenBOQ, which creates multiple products using a total development budget; maximizing total profitability in this case is very challenging.

Acknowledgement

This work was supported by grant HKU 714212E from Hong Kong RGC and grant MYRG109(Y1-L3)-FST12-ULH from University of Macau Research Committee.

A. Compact MBR Computation

Given an MBR m, a budget plane $\mathcal{BP}$, and an orthogonal hyperplane, we can partition m into (at most) two compact MBRs as shown in Figure 8(a). In this section, we show how we can compute the boundaries of these new MBRs in $O(d^2)$ time. For the sake of presentation, we assume that the hyperplane splits m into exactly two MBRs m_1 and m_2. Assume that m is split by hyperplane $x[j] = v$ (i.e., parallel to dimension j at value v). Let $m^l[i]$ ($m^u[i]$) be the lower (upper) bound of m in dimension i. Then, two *loose* MBRs, whose union equals m can be defined by the following equations.

$$m_1^l := m^l \quad m_1^u[i] := \begin{cases} m^u[i] & \text{if } i \neq j \\ v & \text{if } i = j \end{cases} \tag{6}$$

$$m_2^l[i] := \begin{cases} m^l[i] & \text{if } i \neq j \\ v & \text{if } i = j \end{cases} \quad m_2^u := m^u \tag{7}$$

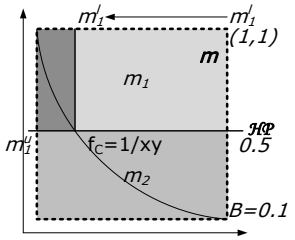


Fig. 19. Compact MBR Computation

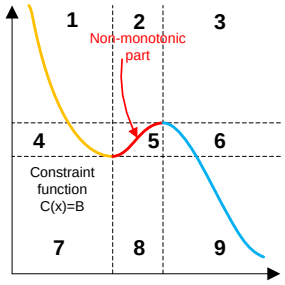


Fig. 20. An example of piecewise functions

Note that the values of m_1^u and m_2^l are the tightest values already; in addition, $m_1^l[j]$ and $m_2^u[j]$ are also tightest. However, we can tighten the values (m_1^l and m_2^u) in other dimensions, so that m_1 and m_2 tightly enclose $\mathcal{BP}$ using the following method. To find the maximum bound within an MBR in dimension i, we need to set the minimum values in all other dimensions and solve $C(x) = B$. For example, $m_1^l[i]$ can be computed by solving the equation $C(x) = B$, by setting $x[j] = m_1^u[j], \forall j \neq i$.

For instance, in Figure 19, assume that we want to partition m using hyperplane y$= 0.5$. Initially, m is split into two MBRs $m_1 = \{m_1^u, m_1^l\} = \{[0.1, 0.5], [1, 1]\}$ and $m_2 = \{m_2^u, m_2^l\} = \{[0.1, 0.1], [1, 0.5]\}$. Then, if $C = \frac{1}{xy}$ and $B = 0.1$, we tighten $m_1^l[0]$ by solving $m_1^l[0] \cdot m_1^u[1] = B$, which gives $m_1^l[0] = 0.1/0.5 = 0.2$. Therefore, the tightmost bound of m_1 becomes $\{[0.1, 0.5], [0.2, 1]\}$. In general, $d - 1$ boundaries can be tightened for each new MBR and each tighening requires $O(d)$ computations, so the overall cost for tightening an MBR is $O(d^2)$.

B. Solution of piecewise constraint functions

Even though our work does not address all non-monotonic constraint functions, our proposed techniques are able to compute GenBOQ if the constraint function $C(x)$ is a *piecewise function* which is composed of a set of finite monotonic functions. For instance,

$$C(x) = \begin{cases} 1/x & 0.5 < x \leq 1 \\ 1 - 1/x & 0 \leq x \leq 0.5 \end{cases} \tag{8}$$

The complete piecewise function may be non-monotonic. Figure 20 illustrates a non-monotonic piecewise function that composes of three finite monotonic functions. In case of such a function, we can partition the domain area into 9 sub-areas. To compute

the result of GenBOQ, we first identify the relevant sub-areas of each finite monotonic function. For instance, the relevant areas of the red function are $2, 3, 4, 5, 6, 7$, and 8 since there is no profitable region of the red function dominating or dominated by areas 1 and 9. Accordingly, we apply our algorithms to compute the GenBOQ result of the red constraint function using the data in the relevant areas. The final result can be straightforwardly produced by combining the result of each piecewise monotonic function.

References

Beckmann, N., Kriegel, H.-P., Schneider, R. and Seeger, B. (1990), The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles, *in* 'SIGMOD Conference', pp. 322–331.

Börzsönyi, S., Kossmann, D. and Stocker, K. (2001), The Skyline Operator, *in* 'ICDE', pp. 421–430.

Chomicki, J., Godfrey, P., Gryz, J. and Liang, D. (2003), Skyline with Presorting, *in* 'ICDE', pp. 717–816.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009), *Introduction to Algorithms, Third Edition*, The MIT Press.

Feng, H., Song, G., Zheng, Y. and Xia, J. (2003), A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Dependent Tasks in Grid Computing, *in* 'GCC (2)', pp. 113–120.

Household dataset (2008). `http://www.ipums.org/`.

Kossmann, D., Ramsak, F. and Rost, S. (2002), Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, *in* 'VLDB', pp. 275–286.

Kung, H. T., Luccio, F. and Preparata, F. P. (1975), 'On Finding the Maxima of a Set of Vectors', *J. ACM* **22**(4), 469–476.

Li, C., Ooi, B. C., Tung, A. K. H. and Wang, S. (2006), DADA: A Data Cube for Dominant Relationship Analysis, *in* 'SIGMOD Conference', pp. 659–670.

Li, C., Tung, A. K. H., Jin, W. and Ester, M. (2007), On Dominating Your Neighborhood Profitably, *in* 'VLDB', pp. 818–829.

Lin, C.-Y., Koh, J.-L. and Chen, A. L. (2012), 'Determining *k*-Most Demanding Products with Maximum Expected Number of Total Customers', *IEEE Transactions on Knowledge and Data Engineering* **PrePrints**(99), 1.

Lu, H. and Jensen, C. S. (2012), Upgrading Uncompetitive Products Economically, *in* 'ICDE', pp. 977–988.

Miah, M., Das, G., Hristidis, V. and Mannila, H. (2008), Standing Out in a Crowd: Selecting Attributes for Maximum Visibility, *in* 'ICDE', pp. 356–365.

NBA Basketball Statistics (2009). NBA Basketball Statistics `http://www.databasebasketball.com/`.

Oliver, D. (2010), 'JoBS Methods Descriptions for the Scientific Study of Basketball', Website. `http://www.powerbasketball.com/theywin2.html`.

Papadias, D., Tao, Y., Fu, G. and Seeger, B. (2005), 'Progressive Skyline Computation in Database Systems', *ACM Trans. Database Syst.* **30**(1), 41–82.

Papadopoulos, A. N., Lyritsis, A., Nanopoulos, A. and Manolopoulos, Y. (2007), Domination Mining and Querying, *in* 'DaWaK', pp. 145–156.

Peng, Y., Wong, R. C.-W. and Wan, Q. (2012), 'Finding Top-k Preferable Products', *IEEE Trans. Knowl. Data Eng.* **24**(10), 1774–1788.

Pro Basketball in USAToday (2009). `http://content.usatoday.com/sports/basketball/nba/salaries/default.aspx`.

Pujowidianto, N. A., Lee, L. H., Chen, C.-H. and Yap, C. M. (2009), Optimal Computing Budget Allocation for Constrained Optimization, *in* 'Winter Simulation Conference', pp. 584–589.

Tan, K.-L., Eng, P.-K. and Ooi, B. C. (2001), Efficient Progressive Skyline Computation, *in* 'VLDB', pp. 301–310.

Wan, Q., Wong, R. C.-W., Ilyas, I. F., Özsu, M. T. and Peng, Y. (2009), 'Creating Competitive Products', *PVLDB* **2**(1), 898–909.

Wan, Q., Wong, R. C.-W. and Peng, Y. (2011), Finding Top-k Profitable Products, *in* 'ICDE', pp. 1055–1066.

Wu, T., Sun, Y., Li, C. and Han, J. (2010), Region-Based Online Promotion Analysis, *in* 'EDBT', pp. 63–74.

Wu, T., Xin, D., Mei, Q. and Han, J. (2009), 'Promotion Analysis in Multi-Dimensional Space', *PVLDB* **2**(1), 109–120.

Yang, Z., Li, L. and Kitsuregawa, M. (2008), Efficient Querying Relaxed Dominant Relationship between Product Items Based on Rank Aggregation, *in* 'AAAI', pp. 1261–1266.

Yang, Z., Wang, B. and Kitsuregawa, M. (2007), General Dominant Relationship Analysis Based on Partial Order Models, *in* 'SAC', pp. 470–474.
Yiu, M. L. and Mamoulis, N. (2009), 'Multi-dimensional Top-*k* Dominating Queries', *VLDB J.* **18**(3), 695–718.
Zhang, S., Mamoulis, N. and Cheung, D. W. (2009), Scalable Skyline Computation Using Object-based Space Partitioning, *in* 'SIGMOD Conference', pp. 483–494.
Zhang, Y., Jia, Y. and Jin, W. (2011), Promotional Subspace Mining with EProbe Framework, *in* 'CIKM', pp. 2185–2188.
Zhang, Z., Lakshmanan, L. V. S. and Tung, A. K. H. (2009), 'On Domination Game Analysis for Microeconomic Data Mining', *TKDD* **2**(4).
Zhou, Y., Chakrabarty, D. and Lukose, R. M. (2008), Budget Constrained Bidding in Keyword Auctions and Online Knapsack Problems, *in* 'WWW', pp. 1243–1244.
Zhu, L., Li, C., Tung, A. K. H. and Wang, S. (2012), 'Microeconomic Analysis Using Dominant Relationship Analysis', *Knowl. Inf. Syst.* **30**(1), 179–211.

Author Biographies

Shen Ge received his Bachelor's and Master's Degree in Computer Science from the Department of Computer Science and Technology in Nanjing University, China, in 2005 and 2008, respectively. He received his PhD's degree in computer science from the University of Hong Kong in 2013, under the supervision of Prof. Nikos Mamoulis. His research focused on query processing on multidimensional and spatial-textual data.

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

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

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