Noname manuscript No.
(will be inserted by the editor)

Quantifying the Competitiveness of a Dataset in Relation to General Preferences

Kyriakos Mouratidis · Keming Li · Bo Tang

the date of receipt and acceptance should be inserted later

Abstract Typically, a specific market (e.g., of hotels, restaurants, laptops, etc) is represented as a multi-attribute dataset of the available products. The topic of identifying and shortlisting the products of most interest to a user has been well-explored. In contrast, in this work we focus on the dataset, and aim to assess its competitiveness with regard to different possible preferences. We define measures of competitiveness, and represent them in the form of a heat-map in the domain of preferences. Our work finds application in market analysis and in business development. These applications are further enhanced when the competitiveness heat-map is used in tandem with information on user preferences (which can be readily derived by existing methods). Interestingly, our study also finds side-applications with strong practical relevance in the area of multi-objective querying. We propose a suite of algorithms to efficiently produce the heat-map, and conduct case studies and an empirical evaluation to demonstrate the practicality of our work.

K. Mouratidis
School of Computing and Information Systems, Singapore Management University
E-mail: kyriakos@smu.edu.sg

K. Li
Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology
E-mail: likm2020@mail.sustech.edu.cn

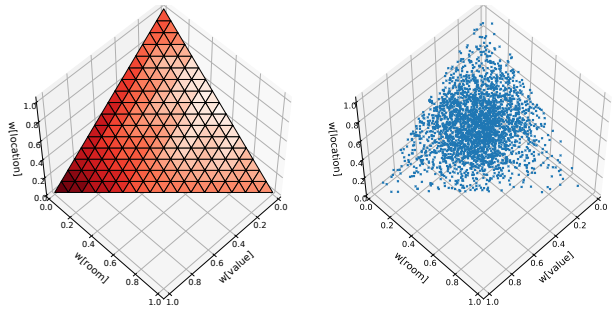
B. Tang
Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology
E-mail: tangb3@sustech.edu.cn

1 Introduction

In a market where the available products are evaluated based on multiple aspects, the suitability of a product and/or its superiority over a competing product is a subjective matter that depends on the preferences of each individual user. Let $\mathcal{P}$ be the dataset of products available in the market to satisfy a specific need (e.g., hotels in New York City), where each product is represented by d attributes (e.g., TripAdvisor ratings on service, cleanliness, value, etc). There has been extensive research on multi-objective operators to identify the products in $\mathcal{P}$ of highest interest to a user, such as the top-k query [37,73], the skyline [15,55], and hybrids of the two [19,48]. While the topic of shortlisting products for a user has been well-explored, in this paper we focus on the dataset itself, aiming to quantify and analyze its competitiveness.

Competitiveness as a general term has been used in the preference querying literature. However, all its past definitions (i) apply to a single product $\boldsymbol{p}$ in isolation, and (ii) are defined according to how many products $\boldsymbol{p}$ dominates or is dominated by [43,79,91,80], or how many top-k queries (from a known and given set) include it in their result [47,78, 58]. In contrast, we aim for a holistic representation of the competitiveness of *the entire dataset*, broken down *per user type* for any kind of possible preferences. To our knowledge, no past work considers competitiveness under that prism. Importantly, the applications of our work are also very different, revolving around the *overall competitive landscape* as opposed to the impact of an isolated product. To make the applications as concrete as possible, we will elaborate on them only after our general problem definition, with reference to a case study we conducted on real data.

To model preferences, we follow the most proliferate [39,23] and effective [59] representation by numerical weights in a linear utility model. That is, each preference



(a) Competitiveness heat-map (b) Actual user preferences

Fig. 1 Heat-map for TripAdvisor hotels, and actual user preferences

profile is characterized by a d-dimensional vector $\boldsymbol{w}$, comprising one weight per product attribute. We refer to the domain of that vector as the *user spectrum*, and define competitiveness in the same domain. Our competitiveness measures draw directly from the principles that underlie standard multi-objective querying. In particular, they fall under two categories, i.e., *utility-based* (which aim to capture how satisfied the different user types are expected to be with the products available in $\mathcal{P}$) and *competition-based* (which aim to quantify how steep the competition among alternative products is with regard to different preferences). Specified a competitiveness measure, we partition the user spectrum into cells. Our computational task is to associate every cell with a competitiveness value for the chosen measure. Together, the competitiveness-marked cells form a heat-map that represents how competitive the dataset is at different parts of the user spectrum, i.e., for different types of users.

To exemplify, Figure 1(a) presents the (utility-based) heat-map for a real dataset of 1,850 hotels, using as attributes their actual TripAdvisor ratings on value, room, and location [5]. The *preference weights* on these three aspects define a triangular user spectrum. We represent the heat-map in that spectrum. The lighter (deeper) the color of a cell, the lower (higher, respectively) its competitiveness. While the case study specifics are elaborated in a dedicated section, the example demonstrates how the heat-map can be used for market analysis and for the identification of opportunities for business development.

Application 1: Market Analysis. On this front, the cells with the highest competitiveness values indicate where the market's strength lies and what types of customers have attracted most of its efforts. This may lend a better understanding of supply dynamics or spark further investigation of the reasons behind the high competitiveness (or lack thereof) for certain types of users. In our example, the deepest-colored cells concentrate around the left corner of the spectrum, meaning that *the hotel market caters best for (and, thus, has likely been focusing its efforts and resources to serve) users who prioritize value over room quality and location*.

Application 2: Business Development. On the front of business development, high competitiveness cells may indicate a saturation of the market for the respective parts of the user spectrum. Conversely, low competitiveness regions could indicate sweet spots for the introduction of a new product $\boldsymbol{p}$ (e.g., the setup of a new hotel). Our heat-map in Figure 1(a) suggests that it may be *easier to introduce a high-ranking hotel when it is tailored to preferences in the lighter-colored cells*. Importantly, we note that after the target cells (preference regions) in the user spectrum are identified, there already exist methods to determine what attributes $\boldsymbol{p}$ should have so that it ranks, say, among the top-k for any [71] or for a desired fraction [89, 88, 69] of the users in those regions.

When the distribution (or a representative sample) of the user preferences is known, the heat-map enables even stronger support in both aforementioned applications. There is a variety of proven preference learning approaches, specifically for the linear utility model, which enable the mining of actual preference samples/distribution, like those surveyed in [28]. In our TripAdvisor case study, we used the preference learning technique in [81] to extract 137,563 weight vectors from the text and scores in actual user reviews on the same portal. Figure 1(b) demonstrates those vectors in the 3-dimensional user spectrum.[1] Observe that our heat-map in Figure 1(a) represents the competitiveness distribution of $\mathcal{P}$ (as a discrete multivariate function) *in the same domain*, i.e., in the user spectrum. The similarity between the two distributions (Figures 1(a) and 1(b)) indicates how well the market's strength is aligned with actual user demand; to assess the degree of alignment, we may use standard measures for distribution similarity, such as the Bhattacharyya coefficient [14], the Jensen–Shannon divergence [44], etc. In our scenario, for instance, the Bhattacharyya coefficient (which takes values between 0 and 1, the higher the more similar) is 0.36. This modest alignment may indicate that *there is business potential and space for further market customization to user demand*. At the same time, availability of the user preference distribution may also complement our heat-map in choosing the target cells (regions in the user spectrum) that a new hotel $\boldsymbol{p}$ should aim for. For example, the competitiveness heat-map and the preference distribution could be combined into a joint, bi-criteria optimization search in the user spectrum to identify its most promising parts (cells) for business growth.

In addition to its main applications, our processing framework has side-applications with strong practical relevance, such as the identification of outstanding products in the market, and top-k query acceleration.

Identifying Outstanding Products. Our utility-based heat-map represents the utility "fringe" of the dataset at dif-

[1] For legibility, only a 2% random sample of the weight vectors is illustrated.

ferent parts of the user spectrum. If a product's utility stands out significantly from that forefront, it is likely an exceptional product. In Section 6, we describe a concrete methodology for the identification of outstanding products, with the adaptation of established outlier detection measures.

Top-k Acceleration. Our processing framework does not require any form of pre-computation, other than the existence of a general purpose index on $\mathcal{P}$. For the case of static product sets, however, our heat-map itself could be used as a materialized structure to expedite the processing of top-k queries on $\mathcal{P}$. As we explain in Section 6, the heat-map may serve as a look-up table that reduces top-k search to plain range search, drastically shortening the computation time.

Our contributions can be summarized as follows:

- We propose a new problem with applications in the analysis of a market and the identification of opportunities in it.
- We formalize the problem based on the practical needs and the principles that underlie multi-objective querying.
- We develop a comprehensive suite of algorithms for its efficient processing.
- We identify promising byproducts of our study that have their own merit.
- We demonstrate the practicality of our work via (i) case studies with real data, and (ii) a performance evaluation using real and synthetic benchmarks.

The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 formalizes the problem and substantiates the practical rationale behind the specific formalization. Section 4 presents (a baseline and) four novel algorithms to produce the heat-map. Section 6 describes two side-applications of our work. Section 7 presents our case studies, followed by performance experiments. Finally, Section 8 concludes the paper.

2 Related Work

We first survey the two traditional multi-objective queries in databases, together with their most relevant variants. Then, we review a recent and effective concept that hybridizes these two approaches. Next, we discuss an existing type of heat-map for a different problem, and conclude the section with other related studies.

2.1 Traditional Multi-objective Queries

Identifying the products of possible interest from a multi-attribute dataset $\mathcal{P}$ has been well-explored in databases, with the *skyline* and the *top-k* query emerging as the standard operators for this task. Regarding the former, a product $\boldsymbol{p}_i$ is said to dominate another $\boldsymbol{p}_j$ when it is at least as preferable on all aspects, and strictly more preferable on at least one aspect [15]. The skyline of a dataset $\mathcal{P}$ includes those products that are not dominated. The efficient computation of the skyline has been studied extensively in the external memory model, be it with [55] or without an index on $\mathcal{P}$ [61,65]; the interested reader may refer to surveys on skyline processing [34,38]. When $\mathcal{P}$ is indexed, the state of the art algorithm, called BBS [55], uses a search heap to visit index nodes and products in increasing distance from the *top-corner* (i.e., the corner of the product space with the maximum coordinates). BBS maintains an interim skyline set, which is updated as new products are popped from the heap. Encountered nodes and products that are dominated by an existing skyline member are pruned, i.e., disregarded. The skyline is finalized when the heap becomes empty. The *k-skyband* is a generalization of the skyline, which includes all products in $\mathcal{P}$ that are dominated by fewer than k others.

The other traditional approach is the *top-k* query [16,35, 37]. It receives as input a user-specific vector $\boldsymbol{w}$ comprising d weights (one per product dimension), and defines the utility of a product $\boldsymbol{p} \in \mathcal{P}$ as the weighted sum of its attributes, i.e., the dot product $\boldsymbol{w} \cdot \boldsymbol{p}$. The top-$k$ result includes the k products with the highest utility. Assuming an index on $\mathcal{P}$, Tao et al. [73] propose a branch-and-bound top-k processing algorithm, which uses a search heap to visit index nodes and products in increasing utility order. Interesting variants arise when the weight vector is not specified, but assumed to follow a certain distribution. For example, if the distribution is uniform, Soliman et al. [66] compute the most probable top-k result. Given an ad hoc distribution for $\boldsymbol{w}$, Peng and Wong [58] derive a subset of $\mathcal{P}$ that is most likely to include the top-1 result. Reverse variants of top-k have also attracted considerable interest. Most prominently, given a set of weight vectors and specified a product $\boldsymbol{p} \in \mathcal{P}$, the *reverse top-$k$* query identifies the weight vectors that have $\boldsymbol{p}$ among their top-k products [76]. Its monochomatic version [77,70], instead of specific weight vectors, identifies *regions* where any vector would hold $\boldsymbol{p}$ in its top-k result. Along the same lines, the *reverse k-ranks* query [94] identifies the weight vectors (from a given set) that rank $\boldsymbol{p}$ the highest, or that associate $\boldsymbol{p}$ with the highest utility.

2.2 Restricted Dominance

There have been several approaches to hybridize the skyline/k-skyband with the top-k query (e.g., [45,72,84]), comprehensively surveyed in [32]. The most recent attempt relies on *restricted dominance* (r-dominance), a notion that combines gracefully elements from both paradigms. Considering only weight vectors $\boldsymbol{w}$ in a given region (convex polytope) R, a product $\boldsymbol{p}_i$ r-dominates another product $\boldsymbol{p}_j$ if $\boldsymbol{p}_i$'s utility is at least as high as $\boldsymbol{p}_j$'s for every $\boldsymbol{w} \in R$, and strictly higher for at least one $\boldsymbol{w} \in R$. The products

in $\mathcal{P}$ that are not r-dominated by any other product constitute the r-skyline [19]. As a building block for a more complex operator, it was generalized for $k \geq 1$ in [50], giving rise to the rk-skyband, i.e., the set of products in $\mathcal{P}$ that are r-dominated by fewer than k others. The rk-skyband is guaranteed to include the top-k products for any $\boldsymbol{w} \in R$. The notion of r-dominance has been well-received, sparking approaches for incremental rk-skyband [48], as well as r-skyline computation in distributed settings [20] by extending the classic *threshold algorithm* [26].

Related (conceptually, at least) to r-dominance are the query-dependent forms of dominance considered in spatial databases. For example, in the context of spatial skylines [62,64], given a set of query points, a product $\boldsymbol{p}_i$ spatially dominates another $\boldsymbol{p}_j$ if every query point is closer to $\boldsymbol{p}_i$ than to $\boldsymbol{p}_j$. Emrich et al. [25] propose an optimal decision criterion for spatial pruning of complex objects approximated by minimum bounding rectangles (MBRs). Given three multi-dimensional rectangles A, B, and R, they determine whether for any triplet of points $a \in A$, $b \in B$, and $r \in R$, it holds that $dist(a,r) < dist(b,r)$ for some distance metric. If so, they consider that rectangle A dominates B with respect to R. In computing the number of rectangles that dominate an object, they address the challenge of incorporating into the count the rectangles that only partially dominate the object, and develop strategies to conservatively and progressively estimate that count. Query-dependent versions of dominance have also been considered for routes/paths in multi-attribute road networks [49,42].

2.3 RNN Heat-map

Specified a query location $\boldsymbol{q}$ in the product space, the *reverse nearest neighbor* (RNN) query [41] identifies the products that have $\boldsymbol{q}$ as their nearest neighbor across $\mathcal{P} \cup \{\boldsymbol{q}\}$. These products form the RNN set of $\boldsymbol{q}$. The problem has been addressed in 2-dimensional [67,90], multi-dimensional [74], and arbitrary metric spaces [6,7], as well as in uncertain databases [13] and spatial influence studies [82].

Sun et al. [68] propose the RNN heat-map for 2-dimensional product sets. That is, they partition the product plane, such that every possible query location $\boldsymbol{q}$ in a partition has the same RNN set. Defining the influence of a partition as the cardinality of its RNN set, and coloring the partition according to that cardinality, they produce a heat-map that represents (spatial) influence at any location in the product plane. The setting and the produced heat-map are very different from ours. First, the RNN heat-map is defined in the same domain as the products (not the user spectrum). Second, it measures spatial influence which depends on physical distance (as opposed to competitiveness, which relies on utility and preference relations). In addition to solving a different problem, the work on RNN heat-map assumes that the NN-circle for every product $\boldsymbol{p} \in \mathcal{P}$ (i.e., the circle with center at $\boldsymbol{p}$ and radius equal to the distance between $\boldsymbol{p}$ and its nearest neighbor) is pre-computed. In contrast, other than a general purpose index on $\mathcal{P}$, our algorithms require no pre-computation. Finally, unlike our work, the technique in [68] is limited to two dimensions (which is reasonable, given the intended location-based service applications of the RNN heat-map).

2.4 Other Related Work

Our heat-map is effectively a data-driven partitioning (and "coloring" with competitiveness values) of the user spectrum. The most well-known data-driven partitioning of a domain is the Voronoi diagram [52]; the product space is partitioned into Voronoi cells, where each cell corresponds to a product $\boldsymbol{p} \in \mathcal{P}$, such that any point in the cell has $\boldsymbol{p}$ as its nearest neighbor across $\mathcal{P}$. This property has been used to accelerate nearest neighbor search in Euclidean space [63], road networks [40], and uncertain databases [93]. For instance, Sharifzadeh and Shahabi [63] embed the Voronoi diagram into an R-tree to reduce the I/O complexity of nearest neighbor processing in Euclidean space. Remotely, that is similar to a side-application of our heat-map in accelerating top-k search (Section 6).

Competitiveness can be seen as a discrete elevation surface over the user spectrum, represented as a heat-map. Our task is to produce that surface/heat-map. Instead, Wu et al. [83] define a problem (in the context of computational lead-finding and fact-checking) where an elevation surface is already given, and they seek to find m representative points on that surface. The representatives must satisfy a triple objective: have high elevation, be far from each other (in order to be diverse), and be representative of the elevation in their neighborhood (i.e., have similar elevation to nearby points on the surface). In that work, the elevation surface is given (whereas in our case the competitiveness "surface" is the output) and the aim is to select m representative points on it (our output includes no representatives). Furthermore, the problem in [83] is NP-hard, thus resorting to approximate (greedy) and heuristic solutions (hill climbing and k-median-like approaches).

Studies on *regret-minimizing sets* (RMS) produce an m-sized representative subset $S \subset \mathcal{P}$ for a given integer m, which tries to satisfy as it best can any possible user preference (i.e., weight vector). The original RMS formulation [53] defines the *regret ratio* for a weight vector as the relative difference between the utility of the best product in S and the best product in the entire $\mathcal{P}$. The objective is to minimize the maximum regret ratio for any possible weight vector. Subsequent research has considered the original RMS [86,9], as well as variants, such as k-RMS [17,

8], where the regret ratio is defined based on the utility difference between the best product in S and the top-k-th in $\mathcal{P}$. Regret has also been considered in terms of the rank of products (instead of utility); e.g., the objective in [10] is to compute the minimal subset S which contains at least one of the top-k products for any possible weight vector. RMS and its common variants are NP-hard/NP-complete for $d > 2$ [85, 8], hence research has focused on approximate solutions. The RMS stream of work is fundamentally different from ours. First, our objective is to quantify the competitiveness of the dataset (not to compute a set of representative products). Second, our approaches are all exact (as opposed to approximate for RMS). Third, our technical challenges relate to processing efficiency (whereas in RMS the quality of approximation and the proof of error bounds is a main consideration). Finally, a side-contribution of our work hints to top-k query acceleration, but that is for exact top-k processing (not approximate, like in the RMS line of work).

Finally, measuring the competitiveness of a market has been considered in the areas of economics and business management, albeit in a very different context [18,33,30]. The metrics used in those areas are concerned with the market shares of the largest firms to assess how oligopolistic the market is, the price-cost margins and their relation to competition, whether there is insufficient or excessive entry of products, etc. In contrast, we consider specifically multi-attribute product sets that are being browsed by multi-objective preference queries. To our knowledge, no past work assesses the competitiveness of datasets under that prism.

3 Preliminaries

In this section, we first formalize our problem and provide the practical rationale behind the measures and decisions it entails (in Section 3.1). Then, in Section 3.2, we specify technical problem settings, such as data storage and indexing.

3.1 Formalization and Rationale

Let the dataset $\mathcal{P}$ comprise all products available in the market, where each product $\boldsymbol{p} \in \mathcal{P}$ is associated with d attributes (dimensions), i.e., $\boldsymbol{p} = (p[1], p[2], ..., p[d])$, and $d \geq 2$. Without loss of generality, we assume that the attributes are non-negative and that higher values are preferred on each dimension. A user's preferences are characterized by a *weight vector* $\boldsymbol{w} = (w[1], w[2], ..., w[d])$, where the i-th weight indicates the significance of the i-th product attribute to the user, and takes a real value in $[0, 1]$. This is in line with the ubiquitous linear utility model, where the suitability of a product to the user is defined as the dot product $\boldsymbol{w} \cdot \boldsymbol{p}$, i.e.,

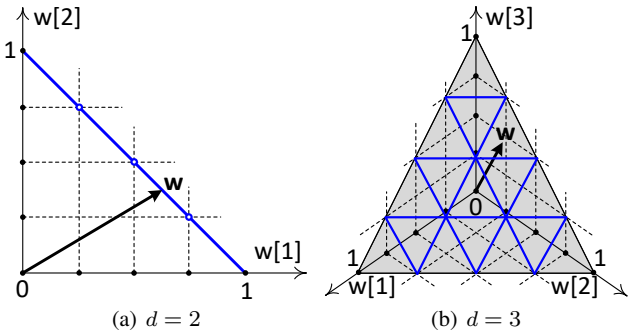


Fig. 2 User spectrum and heat-map cells for $\lambda = 4$

$$\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}) = \boldsymbol{w} \cdot \boldsymbol{p} = \sum_{i=1}^{d} p[i]w[i]$$

This linear type of utility has been the most proliferate since the inception of utility modeling [39,23] and, importantly, it has been shown by user studies to capture effectively the way real users assess the suitability of multi-attribute products [59].

In the above model, the magnitude $|\boldsymbol{w}|$ of the weight vector is unimportant, since $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$ (being the dot product of $\boldsymbol{w}$ and $\boldsymbol{p}$) is directly proportional to $|\boldsymbol{w}|$ for any product [75,36]. We thus follow the common convention that $\sum_{i=1}^{d} w[i] = 1$ for every user/weight vector [16,35,76,66]. That is, the *user spectrum*, i.e., the domain of all possible user preferences, is the simplex defined by $\sum_{i=1}^{d} w[i] = 1$ and $w[i] \in [0, 1], \forall i \in [1, d]$. For $d = 2$, the user spectrum is the diagonal line segment between points $(0, 1)$ and $(1, 0)$ in Figure 2(a). For $d = 3$, it is the shaded triangle in Figure 2(b), etc.

Since utility depends on the weight vector, the competitiveness of the market (i.e., of the dataset) ought to be expressed in relation to each different part of the user spectrum. Thus, we represent competitiveness in the form of a heat-map, which partitions the spectrum into cells. We are to compute a competitiveness value for each cell, for an appropriate competitiveness measure (to be discussed shortly).

To initialize the heat-map, we partition each axis of the spectrum into λ equal ranges; λ is an application-specified parameter that controls the granularity of the heat-map. For ease of presentation, we assume that it is a power of 2. The imposed slicing of the user spectrum produces the heat-map cells. Assuming $\lambda = 4$, Figure 2 demonstrates the 4 cells (segments along the diagonal) produced in $d = 2$, and the 16 cells (triangles in the shaded surface) produced in $d = 3$. Generally, a cell c is the part of the user spectrum bounded by d conditions of the form $L_i \leq w[i] \leq H_i$ for $i \in [1, d]$, where $[L_i, H_i]$ is a range along the $w[i]$ axis. For example,

the cell that contains vector $\boldsymbol{w}$ in Figure 2(b) corresponds to $[0, 0.25] \times [0.25, 0.5] \times [0.5, 075]$.

Having partitioned the user spectrum, we may now "localize" the measuring of competitiveness per cell. Given a cell c, we must first determine which products matter, preference-wise, for weight vectors that lie within c. The standard multi-objective operators, like traditional top-k and k-skyband computation, since their early steps [16,55], have considered more than just the top-utility product, recognizing that a user's decision may also involve several runner-ups. Concordantly, we adopt parameter k from the literature, and address the general $k \geq 1$ case. In particular, we consider that the products which determine competitiveness for preferences in c are those that belong to the rk-skyband for c, i.e., those that are not r-dominated by k or more other products. As adapted from [19,50], r-dominance and rk-skyband are defined as follows.

Definition 1 Given a convex region R in the user spectrum, a product $\boldsymbol{p}_i$ r-dominates another product $\boldsymbol{p}_j$, denoted as $\boldsymbol{p}_i \succ_R \boldsymbol{p}_j$, if and only if $\forall \boldsymbol{w} \in R, \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_i) \geq \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_j)$, and $\exists \boldsymbol{w} \in R, \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_i) > \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_j)$.

Definition 2 Given a positive integer k and a convex region R in the user spectrum, the rk-skyband of dataset $\mathcal{P}$ is the set of products $\boldsymbol{p}_j \in \mathcal{P}$ for which $|\{\boldsymbol{p}_i \in \mathcal{P} : \boldsymbol{p}_i \succ_R \boldsymbol{p}_j\}| < k$.

The reason we adopt this definition is because (i) it accounts for every possible preference in c, (ii) the rk-skyband is guaranteed to hold the top-k products for any weight vector $\boldsymbol{w} \in c$, and (iii) it combines effectively elements from both the standard notions of utility and dominance [19,50, 46]. In the following, we denote the rk-skyband for c as $RKS(c)$.

The next formalization step is to determine intuitive and sensible measures of competitiveness for each cell c in the heat-map. We use the applications described in Introduction as a guide, and introduce two categories of measures:

Utility-based: Given that the utility function captures how well the user preferences are met, probably the most natural indication of competitiveness for c is the utility of products in $RKS(c)$. Assigning a single utility value to c, however, requires appropriate aggregation. First, there are multiple products in $RKS(c)$, and second, even in isolation, the utility $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$ of each of these products varies for different weight vectors in c. While alternative aggregations are possible (and supported by our framework), we choose as most indicative a formulation that aligns best with our intended applications. Specifically, (on the front of market analysis) our measure represents a *for-granted utility* that any of the possible top-k products would have for any preference in c. As such, it also serves (on the front of business development) as a lower bound for the utility a new product should have if it aspires to be among the top-k for some preferences in c. We denote it as $MaxMin_k(c)$, and define it as follows. For each product $\boldsymbol{p} \in RKS(c)$, we compute $\boldsymbol{p}$'s minimum possible utility for any $\boldsymbol{w} \in c$, i.e., $\min_{\boldsymbol{w} \in c} \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$. $MaxMin_k(c)$ is the k-th largest $\min_{\boldsymbol{w} \in c} \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$ value across all $\boldsymbol{p} \in RKS(c)$. Formally, if operator $\max^k(.)$ returns the k-th largest from a set of values, $MaxMin_k(c) = \max^k_{\boldsymbol{p} \in RKS(c)} \min_{\boldsymbol{w} \in c} \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$. Competitiveness aside, $MaxMin_k(c)$ is a guaranteed lower bound of the top-k-th utility in $\mathcal{P}$ for any $\boldsymbol{w} \in c$. This property enables an interesting side-application of our work, as we explain in Section 6.

Competition-based: An alternative approach to define competitiveness, is to capture how intense the competition is *among the candidate products* to serve the users represented by c. Measuring the steepness of competition among products is essential for the understanding/analyzing of market dynamics, but also for determining the target regions in the user spectrum for a new product (e.g., low competition cells). Innately, a large number of products in $RKS(c)$ indicates that there is a large concentration of high-ranking competitors for the same part (cell) of the user spectrum, and vice versa. Therefore, a first measure in this category is the cardinality $|RKS(c)|$. Another related measure is the total number of r-dominance relations between products in $RKS(c)$, denoted as $\#DR(c)$. Formally, $\#DR(c) = |\{(\boldsymbol{p}_i, \boldsymbol{p}_j) \in RKS(c)^2 : \boldsymbol{p}_i \succ_c \boldsymbol{p}_j\}|$. We expect the two measures ($|RKS|$ and $\#DR$) to be correlated, but we defer this investigation to our empirical study.

We consider the described measures of competitiveness as the most natural in multi-objective settings, yet our methodology may support more complex or application-specific alternatives, e.g., definitions that combine $MaxMin_k(c)$ and $|RKS(c)|$ into a single formula. For instance, when looking for opportunities for business growth, cells with both low $MaxMin_k(c)$ and low $|RKS(c)|$ could represent low-hanging fruits in the user spectrum. Specified a competitiveness measure, our problem is defined as follows.

Problem 1 Given a dataset $\mathcal{P}$ that represents the products in the market and parameters k and λ, produce a heat-map of the user spectrum where each cell is associated with its competitiveness value.

3.2 Problem Setting

According to the majority of multi-objective decision support applications [37], and given the dimensionality curse that plagues their usefulness for high d [31,92,51], we focus on moderate dimensionalities (e.g., up to 7-8 dimensions). We consider large datasets, i.e., cases where $\mathcal{P}$ includes a considerable number of products. To deal with the scale of $\mathcal{P}$, we assume a general purpose index on it, such

as a standard R*-tree [11]. Given the typical RAM sizes in commodity machines, we keep the index and the dataset in memory. The main performance determinant is computation time, with memory overhead being an additional consideration.

Other than a general purpose index, our methods rely on no pre-computation. Instead, our algorithms perform all required processing at query time. Should an application impose predicates on product attributes (e.g., hotels with cleanliness rating above a certain threshold), the predicates can first be effected using the index, and our methods be simply executed on the selected part of the index and dataset. By default, however, we assume that Problem 1 takes into account the entire dataset $\mathcal{P}$.

To assist in the following, Table 1 serves as a quick reference for commonly used notation.

Notation	Description
$\mathcal{P}$	Dataset (the set of products in the market)
d	Dimensionality (the number of product attributes)
$\boldsymbol{p}$	A product $\boldsymbol{p} = (p[1], p[2], \cdots, p[d])$ in $\mathcal{P}$
$\boldsymbol{w}$	A weight vector $\boldsymbol{w} = (w[1], w[2], \cdots, w[d])$
$\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$	Utility of product $\boldsymbol{p}$ according to $\boldsymbol{w}$
λ	The heat-map's granularity parameter
c	A heat-map cell (i.e., a region in the user spectrum)
$RKS(c)$	The rk-skyband for cell c
$MaxMin_k(c)$	*MaxMin* k-th utility
$\lvert RKS(c)\rvert$	No. of products in $RKS(c)$
$\#DR(c)$	No. of r-dominance relations in $RKS(c)$
$\overline{RKS}(N)$	Superset of the rk-skyband for node N
$\mathcal{T}(N)$	Threshold for node N
$\mathcal{U}^k_{\boldsymbol{w}}$	Utility of the top-k-th product for $\boldsymbol{w}$

Table 1 Notation

4 Processing Algorithms

Having defined our problem and objectives, our next consideration is efficient processing, i.e., producing the heat-map in reasonable time. This is an important practicality determinant, since our motivating scenarios often entail exploratory analysis and potentially repetitive trials for different parameters (e.g., k, λ, etc).

As elaborated in Section 3.1, for any of the competitiveness measures, the products that matter preference-wise for a cell c are in its rk-skyband. We therefore focus our algorithmic design on the efficient computation of $RKS(c)$ for every cell in the heat-map.

4.1 Baseline

A straightforward approach (denoted as BL) is to execute a standard rk-skyband computation algorithm for every cell. We adopt the algorithm in [50], as it applies directly to general $k \geq 1$. The standard rk-skyband operator was formulated for a convex polytope. Each cell c in the heat-map is such a polytope. The existing r-dominance test reduces to utility comparisons for the extreme vertices of c. Specifically, given two products[2] $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, if $\mathcal{U}_{\boldsymbol{v}}(\boldsymbol{p}_i) \geq \mathcal{U}_{\boldsymbol{v}}(\boldsymbol{p}_j)$ for every extreme vertex $\boldsymbol{v}$ of c, then $\boldsymbol{p}_i$ r-dominates $\boldsymbol{p}_j$. If their utility order is reversed for at least one extreme vertex, then none r-dominates the other.

With $RKS(c)$ in hand, computing $|RKS(c)|$ and $\#DR(c)$ is straightforward. Computing $MaxMin_k(c)$, on the other hand, requires finding for each $\boldsymbol{p} \in RKS(c)$ the minimum $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$ for any $\boldsymbol{w} \in c$. Lemma 1 demonstrates how.

Lemma 1 *Given a product* $\mathbf{p}$ *and a cell* c*, the minimum utility* $\mathcal{U}_{\mathbf{w}}(\mathbf{p})$ *across all* $\mathbf{w} \in c$ *is achieved for one of* c*'s extreme vertices.*

Proof Let c's extreme vertices be $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_m$. From the convexity of c, it follows [12] that any $\boldsymbol{w} \in c$ can be expressed as a linear combination $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i$, where $\sum_{i=1}^{m} \alpha_i = 1$ and $\alpha_i \in [0,1], \forall i \in [1,m]$. Without loss of generality, assume that $\boldsymbol{v}_1 \cdot \boldsymbol{p} \leq \boldsymbol{v}_i \cdot \boldsymbol{p}, \forall i \in [2,m]$. It holds that:

$$\boldsymbol{v}_1 \cdot \boldsymbol{p} = (\sum_{i=1}^{m} \alpha_i) \boldsymbol{v}_1 \cdot \boldsymbol{p} = \alpha_1 \boldsymbol{v}_1 \cdot \boldsymbol{p} + \alpha_2 \boldsymbol{v}_1 \cdot \boldsymbol{p} + ... + \alpha_m \boldsymbol{v}_1 \cdot \boldsymbol{p}$$

$$\leq \alpha_1 \boldsymbol{v}_1 \cdot \boldsymbol{p} + \alpha_2 \boldsymbol{v}_2 \cdot \boldsymbol{p} + ... + \alpha_m \boldsymbol{v}_m \cdot \boldsymbol{p} = \boldsymbol{w} \cdot \boldsymbol{p}$$

In other words, $\mathcal{U}_{\boldsymbol{v}_1}(\boldsymbol{p}) \leq \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$ for every $\boldsymbol{w} \in c$.

Performance-wise, the baseline invokes an rk-skyband algorithm for each and every cell in the heat-map, and is expected to be inefficient (if practical at all) because (i) it is unable to share computations among rk-skyband derivations for different cells, and (ii) even for a single cell c, it needs to consider numerous pairs of products, i.e., to perform numerous r-dominance tests. In particular, letting n be the number of products considered during $RKS(c)$ computation (for a single cell),[3] the specific call of the rk-skyband algorithm performs $O(n^2)$ r-dominance tests.

4.2 Computation Sharing Algorithm

The algorithm we develop in this section cannot resolve the second issue above, but it makes a major leap to address the first, i.e., to effectively share computations among the rk-skyband derivations for different cells. It owes its name to that feature, i.e., *Computation Sharing Algorithm* (CSA).

[2] Without loss of generality, we assume that no pair of products have identical attributes in all dimensions.

[3] rk-skyband computation by [50] follows the general branch-and-bound paradigm using the index on $\mathcal{P}$; n here refers to the number of products popped from its search heap and considered for inclusion into $RKS(c)$.

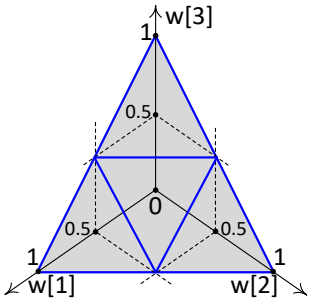


Fig. 3 Split of the root in $d = 3$

4.2.1 Basic CSA

The key idea is that if a product $\boldsymbol{p}_i$ r-dominates another product $\boldsymbol{p}_j$ in cell c, the same is likely to hold in cells that lie near c in the user spectrum. Therefore, instead of performing r-dominance testing per cell, it would be computationally beneficial to cluster hierarchically the cells according to their position in the user spectrum, and determine the r-dominance relation between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ in as high a level in that hierarchy as possible. This way, the r-dominance test at the higher level can be effectively reused in the $RKS(c)$ derivation for any underlying cell.

The hierarchical structure we design, termed *simplex pyramid*, is a balanced tree where every node is a convex region in the user spectrum. The root corresponds to the entire spectrum, while the leaves to the cells of the heat-map. Conceptually, it is formed in a top down manner via recursive splits. Specifically, each internal node N corresponds to the part of the user spectrum bounded by d conditions of the form $L_i \leq w[i] \leq H_i$ for $i \in [1, d]$. N's children are derived by splitting its region according to d hyper-planes, where the i-th hyper-plane is perpendicular to the $w[i]$ axis and has equation $w[i] = \frac{H_i - L_i}{2}$. In other words, we split each $[L_i, H_i]$ range into two. Assuming $d = 3$, Figure 3 demonstrates the split of the root (entire spectrum), which produces 4 children (triangles) shown with bold boundaries. The splitting continues recursively, until all leaves of the pyramid are at depth $\log_2(\lambda)$; the nodes (leaves) at that depth are the cells of the heat-map. Lemma 2 offers insight in the structure of the simplex pyramid, showing that at depth h there are $\binom{2^h-1+d}{d} - \binom{2^h}{d}$ nodes.

Lemma 2 *In general dimensionality d, there are $\binom{2^h-1+d}{d} - \binom{2^h}{d}$ nodes at depth h of the simplex pyramid.*

Proof Without loss of generality, in our proof we consider an open-set representation of the nodes. A node at depth h of the simplex pyramid is expressed as:

$$\begin{cases} \sum_{i=1}^{d} w_i = 1, \\ \frac{n_i}{2^h} < w_i < \frac{n_i+1}{2^h}, \text{where } 0 \leq n_i \leq 2^h - 1, n_i \in \mathbb{N}_0. \end{cases} \quad (1)$$

That is equivalent to:

$$\begin{cases} \sum_{i=1}^{d} \frac{n_i}{2^h} < 1, \\ \sum_{i=1}^{d} \frac{n_i+1}{2^h} > 1. \end{cases} \quad (2)$$

For every $i \in [1, d]$, n_i is a non-negative integer, hence $2^h - (d-1) \leq \sum_{i=1}^{d} n_i \leq 2^h - 1$. Therefore, finding how many nodes are at depth h is the same as finding how many solutions there are in the following system of inequalities:

$$\begin{cases} 2^h - (d-1) \leq \sum_{i=1}^{d} n_i \leq 2^h - 1, \\ 0 \leq n_i \leq 2^h - 1, n_i \in \mathbb{N}_0. \end{cases} \quad (3)$$

Note that the number of solutions to

$$\begin{cases} \sum_{i=1}^{d} n_i = a, \text{where } 0 \leq n_i \leq a, n_i \in \mathbb{N}_0, \\ a \in \mathbb{N}_0 \end{cases} \quad (4)$$

is the number of combinations $\binom{a+d-1}{d-1}$ [27]. Therefore, the number of solutions to equation (3) is $\sum_{i=2^h-(d-1)}^{2^h-1} \binom{i+d-1}{d-1}$. Letting $L = 2^h - (d-1)$ and $H = 2^h - 1$, and keeping in mind the property $\binom{i+d-1}{d-1} + \binom{i+d-1}{d} = \binom{i+d}{d}$, the number of solutions (and therefore the number of nodes at depth h) is:

$$\begin{aligned}
&\textstyle\sum_{i=2^h-(d-1)}^{2^h-1} \binom{i+d-1}{d-1} = \sum_{i=L}^{H} \binom{i+d-1}{d-1} \\
&= \left[\textstyle\sum_{i=L}^{H} \binom{i+d-1}{d-1}\right] + \binom{L+d-1}{d} - \binom{L+d-1}{d} \\
&= \left[\textstyle\sum_{i=L+1}^{H} \binom{i+d-1}{d-1}\right] + \binom{L+d-1}{d-1} + \binom{L+d-1}{d} - \binom{L+d-1}{d} \\
&= \left[\textstyle\sum_{i=L+1}^{H} \binom{i+d-1}{d-1}\right] + \binom{L+d}{d} - \binom{L+d-1}{d} \\
&= \left[\textstyle\sum_{i=L+2}^{H} \binom{i+d-1}{d-1}\right] + \binom{L+d}{d-1} + \binom{L+d}{d} - \binom{L+d-1}{d} \\
&= \left[\textstyle\sum_{i=L+2}^{H} \binom{i+d-1}{d-1}\right] + \binom{L+d+1}{d} - \binom{L+d-1}{d} \\
&\vdots \\
&= \binom{H+d}{d} - \binom{L+d-1}{d} = \binom{2^h-1+d}{d} - \binom{2^h}{d},
\end{aligned}$$

which completes the proof.

A direct corollary that follows from Lemma 2 for $h = \log_2(\lambda)$, is that the total number of cells in the heat-map is $\binom{\lambda-1+d}{d} - \binom{\lambda}{d}$.

After initializing an empty simplex pyramid, CSA computes the traditional k-skyband of $\mathcal{P}$ using some standard algorithm, e.g., BBS [55]. By definition, the rk-skyband for any region in the user spectrum is a subset of the k-skyband, i.e., members of the latter are the only products we need to consider henceforth. For each of these products $\boldsymbol{p}$, CSA computes the *dominance count* (i.e., the number of other products that dominate $\boldsymbol{p}$), and records all counts in an array. That array is stored in the root of the pyramid (as a constant), and replicated into each of the other nodes (as an initialization, to be updated later).

In the next phase, CSA considers every pair of products $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ from the k-skyband that do not (traditionally) dominate each other, and inserts these pairs into the pyramid one

by one. To insert a pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$, CSA traverses the pyramid in a top-down fashion, starting with the root. For every child node N of the root, we perform the standard r-dominance test described in Section 4.1. If one of the products in the pair, say, $\boldsymbol{p}_j$ is found to be r-dominated by the other (i.e., by $\boldsymbol{p}_i$), we increment the dominance count of $\boldsymbol{p}_j$ in the array of N *and in the arrays of all nodes in the sub-tree under* N, *without any r-dominance testing*.[4] In contrast, if the test determines that there is no r-dominance relation between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, no count increment is made in N's array, and the insertion routine is invoked recursively in its children (if N is already a leaf, no recursive call is made). At any point, a product's dominance count in the array of a node N indicates the total number of products found so far to dominate or r-dominate it in N. When all pairs have been inserted, the array in each leaf of the pyramid (i.e., in each cell c of the heat-map) provides $RKS(c)$; specifically, $RKS(c)$ includes those and only those products with dominance count smaller than k.

A useful enhancement is that when the dominance count of a product $\boldsymbol{p}$ in node N reaches k, we deem the product as *eliminated* for N, i.e., $\boldsymbol{p}$ cannot be in the rk-skyband for N or for any node in its sub-tree. Any subsequent invocation of the insertion routine in N for a product pair that includes $\boldsymbol{p}$ will be ignored (no r-dominance testing will take place and no insertion attempt will be made in the descendants of N).

A note about the basic CSA algorithm is that the order it considers the product pairs is not important. The reason is that, given a level of the pyramid, the hyper-planes $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_i) = \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_j)$ defined for each of the pairs $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ are expected to cut through a similar number of nodes in that level, as suggested by the zone theorem [24]; given an arrangement of n hyper-planes in a d-dimensional space, and another, new hyper-plane in the same space, the zone theorem states that the latter hyper-plane cuts through $O(n^{d-1})$ cells of the arrangement. It follows that the number of nodes that lie completely on either side of these hyper-planes (i.e., the nodes where one of the products is found to r-dominate the other) is similar too. Therefore, the number of dominance count increments per level (and, by extension, in the entire pyramid) is expected to be similar for every product pair considered.

4.2.2 Extended CSA

Here we present CSA$^+$, an extended version of CSA that aims to reduce the number of r-dominance tests by exploiting the transitivity of the dominance/r-dominance relations.

CSA$^+$ initializes an empty simplex pyramid and starts, as per normal, by computing the k-skyband of $\mathcal{P}$. However, at that stage, it also computes the *dominance graph*, Γ; that is a directed acyclic graph, which represents every (traditional) dominance relation as a directed edge from the dominating product to the dominated one. Every node N in the pyramid implicitly inherits Γ from the root, but it additionally keeps a *delta-graph*, $\Delta(N)$, which records (in the form of directed edges) all the r-dominance relations in N discovered during CSA$^+$ execution. A product's sum of in-degrees in Γ and in $\Delta(N)$ is the product's dominance count in N, as defined previously for CSA.

Inserting a product pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ in a node N is the same as in CSA. However, when a product in the pair, say $\boldsymbol{p}_i$, r-dominates the other (i.e., $\boldsymbol{p}_j$), we do not only add an edge in $\Delta(N)$ from $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, but also from $\boldsymbol{p}_i$ to any node $\boldsymbol{p}_o$ that is dominated or r-dominated by $\boldsymbol{p}_j$ (as captured in Γ and $\Delta(N)$, respectively). This is the main enhancement of CSA$^+$ over CSA, which helps determine some r-dominance relations (thus increase dominance counts and, in turn, bring products closer to elimination for N) without r-dominance testing. Every new edge added to $\Delta(N)$ is also copied to the delta-graphs in the sub-tree under N in the pyramid.

Extra care is required in order to properly exploit transitivity. CSA$^+$ may add edges, like $\boldsymbol{p}_i \rightarrow \boldsymbol{p}_o$ in our example, without yet having considered the pair $(\boldsymbol{p}_i, \boldsymbol{p}_o)$. Therefore, before inserting any product pair in a node N, we need to look into $\Delta(N)$ and ignore the pair (for N and its sub-tree) if there is already an edge between the specific two products. As per usual, we also ignore the pair if either of the products already has a dominance count of k.

Unlike CSA, in CSA$^+$ the order that pairs are considered does matter, exactly because it aims to exploit the transitivity of dominance/r-dominance relations. Intuitively, products with small (traditional) dominance counts in Γ are likely to r-dominate many others. Thus, we order the products in increasing (traditional) dominance count and form pairs between the first product and each of the subsequent products in the order, then between the second product and all its succeeding products, and so on.

Algorithm 1 summarizes the CSA$^+$ process. After sorting in Line 4, we assume that $\boldsymbol{p}_i$ refers to the i-th product in the imposed order. Regarding Lines 10 and 16, recall that a product's dominance count in a node N is the sum of the product's in-degrees in Γ and in $\Delta(N)$.

4.3 Mapping-based Dominance Algorithm

CSA and CSA$^+$ insert into the pyramid *every pair of products* from the k-skyband that do not dominate each other, i.e., letting n be the cardinality of the k-skyband, they insert $O(n^2)$ pairs. The *mapping-based dominance algorithm* (MDA) removes this combinatorial requirement by largely individualizing the assessment of products. Specifically, for the most part, it abandons the pair-based r-dominance tests;

[4] If $\boldsymbol{p}_i$ r-dominates $\boldsymbol{p}_j$ for N, the same holds for every partition of N too.

Algorithm 1 CSA$^+(\mathcal{P}, k, \lambda)$

```
1: Initialize the simplex pyramid according to λ
2: S ← the k-skyband of P
3: Compute the dominance graph Γ for products in S
4: Sort the products in S in increasing dominance count order
5: for each (p_i, p_j) ∈ S^2, i < j and p_i does not dominate p_j do
6:     Insert(p_i, p_j, N)                    ▷ Insert pair (p_i, p_j) in node N
7: for each leaf c in the pyramid do          ▷ Get RKS(c) using Δ(c)
8:     Initialize empty RKS(c)
9:     for each p ∈ S do
10:        if the in-degree of p in Δ(c) ∪ Γ is < k then
11:            Insert p into RKS(c)
12:    Compute competitiveness for c based on RKS(c)

13: Routine Insert(p_i, p_j, N)
14: if edge p_i → p_j or p_j → p_i is already in Δ(N) then
15:     Return
16: if the in-degree of p_i or p_j in Δ(N) ∪ Γ is ≥ k then
17:     Return
18: if p_i r-dominates p_j in N then
19:     Add p_i → p_j to Δ(N)
20:     for each p_o such that p_j → p_o ∈ Δ(N) ∪ Γ do
21:         Add p_i → p_o to Δ(N)
22:     Add edges in Lines 19 and 21 to delta-graphs in N's sub-tree
23: else if p_j r-dominates p_i in N then
24:     Handle similarly to Lines 19-22 with p_j as dominating product
25: else                                      ▷ No r-dominance between p_i and p_j in N
26:     for each child of node N do
27:         Insert(p_i, p_j, N.child)
```

instead, it quantifies the potential of a product $\boldsymbol{p}$ to enter the rk-skyband in more absolute terms.

Given a product $\boldsymbol{p}$ and a node N in the simplex pyramid, MDA builds on the minimum and the maximum possible utility that $\boldsymbol{p}$ may achieve for any $\boldsymbol{w} \in N$; we denote these values as $LU_N(\boldsymbol{p})$ and $HU_N(\boldsymbol{p})$, respectively. $LU_N(\boldsymbol{p})$ is computed by applying Lemma 1 to node N instead of a cell c (recall that, like c, every node N is a convex polytope). $HU_N(\boldsymbol{p})$, on the other hand, is equal to $\max_{i\in[1,m]} \mathcal{U}_{\boldsymbol{v}_i}(\boldsymbol{p})$, where $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_m$ are the extreme vertices of (the polytope that corresponds to) N, as follows from Lemma 3.

Lemma 3 *Given a product* $\mathbf{p}$ *and a node* N *in the simplex pyramid, the maximum utility* $\mathcal{U}_{\mathbf{w}}(\mathbf{p})$ *across all* $\mathbf{w} \in N$ *is achieved for one of* N*'s extreme vertices.*

Proof The proof is identical to Lemma 1, subject to replacing '$\leq$' with '$\geq$', and c with N.

This way, any product can be mapped to a utility range $[LU_N(\boldsymbol{p}), HU_N(\boldsymbol{p})]$ for node N. If for two products $\boldsymbol{p}_i, \boldsymbol{p}_j$ it holds that $LU_N(\boldsymbol{p}_i) \geq HU_N(\boldsymbol{p}_j)$, product $\boldsymbol{p}_i$ is guaranteed to r-dominate $\boldsymbol{p}_j$ in N. That said, note that this condition is sufficient but not necessary, i.e., it is still possible for a product to r-dominate another, although their utility ranges overlap. Figure 4 demonstrates such a situation in $d = 2$, showing the utility of two products, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, in a node N. To be able to visualize, we use the ratio $\frac{w[1]}{w[2]}$ to express

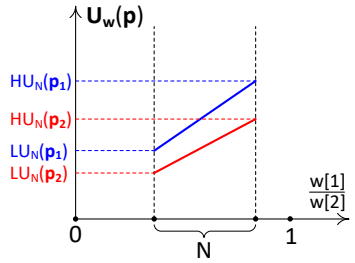


Fig. 4 Range overlap, although $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_1) \geq \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_2), \forall \boldsymbol{w} \in N$

the user spectrum (horizontal axis), which is an alternative, yet equivalent representation of the diagonal in Figure 2(a); every weight vector $\boldsymbol{w}$ on the diagonal has a one-to-one correspondence with a ratio $\frac{w[1]}{w[2]}$, i.e., with a vertical line in Figure 4. Although the utility ranges of $\boldsymbol{p}_1, \boldsymbol{p}_2$ overlap, for every $\boldsymbol{w} \in N$ (i.e., for every vertical line inside N) the utility of $\boldsymbol{p}_1$ is higher, thus it r-dominates $\boldsymbol{p}_2$ in N.

Nonetheless, while comparing utility ranges is not a panacea, observe that a product's range for N is fixed and irrelevant to other products. That is, range $[LU_N(\boldsymbol{p}), HU_N(\boldsymbol{p})]$ may be computed once and used directly anytime $\boldsymbol{p}$ needs to be compared against other products for N.

4.3.1 Basic MDA

The main idea in MDA is to employ a simplex pyramid built on top of the heat-map, but *to insert individual products* into it, as opposed to pairs of products. In every node N of the pyramid, MDA maintains *a superset of the rk-skyband for* N (over the products inserted so far), denoted as $\overline{RKS}(N)$. The k-th largest $LU_N(\boldsymbol{p})$ value in $\overline{RKS}(N)$ is denoted as $\mathcal{T}(N)$ and serves as an admission threshold to $\overline{RKS}(N)$. When the insertion routine is invoked in N for a product $\boldsymbol{p}$, MDA computes $LU_N(\boldsymbol{p})$ and $HU_N(\boldsymbol{p})$. If $HU_N(\boldsymbol{p})$ is no larger than $\mathcal{T}(N)$, $\boldsymbol{p}$ is guaranteed to be r-dominated by at least k products in $\overline{RKS}(N)$, and hence it cannot be in the rk-skyband for N or for any node in N's sub-tree; $\boldsymbol{p}$ is eliminated for N and no insertion is attempted in N's children. On the other hand, if $HU_N(\boldsymbol{p}) > \mathcal{T}(N)$, we (i) include $\boldsymbol{p}$ in $\overline{RKS}(N)$, (ii) update $\mathcal{T}(N)$, and (iii) if N is an internal node, invoke the insertion routine for $\boldsymbol{p}$ in N's children.

MDA inserts the k-skyband products into the simplex pyramid one by one. The recursive insertion routine is not invoked at the root (whose rk-skyband is anyway the full k-skyband), but directly in its children. After the last product is inserted, the $\overline{RKS}(c)$ at each cell (i.e., at each leaf of the pyramid) is guaranteed to be a superset of $RKS(c)$. However, we may tighten it further. We go through $\overline{RKS}(c)$ and remove products $\boldsymbol{p}$ where $HU_c(\boldsymbol{p}) \leq \mathcal{T}(c)$. That is because $\mathcal{T}(c)$ might have increased since $\boldsymbol{p}$ was first included in $\overline{RKS}(c)$ (due to subsequent insertions).

To complete the process, we derive $RKS(c)$ for every cell c, by considering only the products in $\overline{RKS}(c)$; it is only at this stage that we perform traditional r-dominance testing. To juxtapose that last step with the baseline in Section 4.1, note that $\overline{RKS}(c)$ is expected to be a tight superset of $RKS(c)$, and therefore much smaller than the entire k-skyband. We perform this last step as follows.

Consider a cell c. We initialize an empty $RKS(c)$ set and examine the products in $\overline{RKS}(c)$ in decreasing $HU_c(\boldsymbol{p})$ order. Note that $HU_c(\boldsymbol{p})$ is not just an upper bound, but an actual utility that $\boldsymbol{p}$ achieves for some $\boldsymbol{w} \in c$. Therefore, if for two products $HU_c(\boldsymbol{p}_i) > HU_c(\boldsymbol{p}_j)$, it is impossible for $\boldsymbol{p}_j$ to r-dominate $\boldsymbol{p}_i$ in c. This means that (i) the k first products can be directly placed into $RKS(c)$, and (ii) any subsequent product $\boldsymbol{p}$ can only be r-dominated by products preceding it. Thus, it suffices to test $\boldsymbol{p}$ only against the products already in $RKS(c)$; if r-dominated by fewer than k of them, $\boldsymbol{p}$ is inserted in $RKS(c)$ too (and is certain to remain there). When all products in $\overline{RKS}(c)$ have been considered, $RKS(c)$ becomes final.

Crucial point: To offer perspective, a crucial point is that, in effect, CSA and CSA$^+$ maintain the rk-skyband *for every node* in the simplex pyramid (while it is only required for the leaves), thus wasting computations. MDA, on the other hand, replaces the exact rk-skybands for internal nodes N with fast-to-compute supersets (i.e., $\overline{RKS}(N)$) which enable efficient filtering, without compromising correctness. Refinement, i.e., computation of exact rk-skybands, only takes place for the leaves of the pyramid (i.e., for the cells of the heat-map).

4.3.2 Enhanced MDA

Here we present the *Enhanced MDA* (MDA$^+$). An issue with basic MDA is that it inserts into the simplex pyramid, top-down, each and every product from the k-skyband of $\mathcal{P}$. First, the number of these products may be large [31,55]. Second, the insertion order is global and common for all nodes in the pyramid. This means that MDA may invoke the insertion routine in a node N for products that are far from making it into $\overline{RKS}(N)$. Worse yet, even though a product may make it initially into $\overline{RKS}(N)$ (and thus be also recursively inserted in N's children and, potentially, in N's entire sub-tree), its $HU_N(\boldsymbol{p})$ might turn out to be much lower than *the final* $\mathcal{T}(N)$, i.e., had this product been inserted at a later order, it could have been eliminated directly for N and never been propagated to N's sub-tree.

MDA$^+$ individualizes the $\overline{RKS}(N)$ computation process, so as to (i) avoid propagating *any* product with $HU_N(\boldsymbol{p})$ smaller than *the final value* of $\mathcal{T}(N)$ down to N's sub-tree, and (ii) impose a *node-specific* order among inserted products that allows to safely eliminate products in a branch-and-bound fashion, while still deriving a guaranteed (and tight) superset $\overline{RKS}(N)$ of the rk-skyband for N.

Initially, MDA$^+$ computes the k-skyband of $\mathcal{P}$, and relays it to the children of the root in the simplex pyramid. In each of the root's children N, we derive $\overline{RKS}(N)$ based on the k-skyband products (we describe the derivation of $\overline{RKS}(N)$ shortly), and only when $\overline{RKS}(N)$ is finalized, do we relay it to N's own children. In each of N's children, we compute its own $\overline{RKS}$ set based only on the product set that was relayed to it (i.e., $\overline{RKS}(N)$), and so on all the way down to the leaves.

The routine to compute $\overline{RKS}(N)$ in a node N (be it internal or leaf) first sorts the product set relayed to it in decreasing $HU_N(\boldsymbol{p})$, and considers the products in that order. The first k of them are directly inserted in $\overline{RKS}(N)$. We keep inserting in $\overline{RKS}(N)$ the subsequent products $\boldsymbol{p}$ one by one (making sure to update $\mathcal{T}(N)$ along the way) as long as $HU_N(\boldsymbol{p}) > \mathcal{T}(N)$. We stop when we encounter the first product $\boldsymbol{p}$ with $HU_N(\boldsymbol{p}) \leq \mathcal{T}(N)$; $\boldsymbol{p}$ (and all products after it) are unable to enter $\overline{RKS}(N)$, thus $\overline{RKS}(N)$ is now final.

When $\overline{RKS}(c)$ has been derived for all leaves in the simplex pyramid (i.e., for all cells c), we use it to compute the actual $RKS(c)$, following the same technique as the last step in MDA. Algorithm 2 summarizes the whole MDA$^+$ process. Observe that $\overline{RKS}(N)$ computation happens in the recursive Insert() routine, all the way down to the leaves/cells. *Traditional r-dominance testing only takes place at the very end (in Line 10) and only for the leaves.*

In addition to benefits in response time, MDA$^+$ also features lower memory overhead. Since $\overline{RKS}(N)$ computation happens atomically per node, a node's $\overline{RKS}(N)$ can be dismissed once the $\overline{RKS}$ set of all its children has been computed. Actually, with diligent implementation, MDA$^+$ requires the geometric representation and $\overline{RKS}(N)$ set for only $h = \log_2(\lambda)$ nodes at any point during execution (these nodes correspond to a path from the root down to a leaf c). While memory overhead is a secondary performance criterion in our context, it is still a vital aspect that MDA$^+$ has a significant advantage over alternatives, as we show in the experiments.

5 Time and Space Analysis

In this paper, we aim for practical implementations, and thus focus on actual response time (and memory consumption) on real and benchmark data. For completeness, however, in this section we analyze the time and space complexity of the baseline (BL) and of the enhanced algorithms from Sections 4.2 and 4.3 (i.e., CSA$^+$ and MDA$^+$).

Lemma 4 *The time complexity of BL is* $O\left(\left(\frac{k\ln^{d-1}|\mathcal{P}|}{d!}\right)^2 \cdot \left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)\right)$.

Algorithm 2 MDA$^+(\mathcal{P}, k, \lambda)$

```
1: Initialize the simplex pyramid according to λ
2: S ← the k-skyband of P
3: for each child N of the root of the simplex pyramid do
4:     Insert(S, N)                          ▷ Insert product set S in node N
5: for each leaf c in the pyramid do   ▷ Get RKS(c) using RKS̄(c)
6:     Initialize empty RKS(c)
7:     for each p ∈ RKS̄(c) in decreasing HU_c(p) order do
8:         if RKS(c) has fewer than k products then
9:             Insert p into RKS(c)
10:        else if p is dominated by < k products in RKS(c) then
11:            Insert p into RKS(c)
12:    Compute competitiveness for c based on RKS(c)

13: Routine Insert(S, N)
14: Initialize empty RKS̄(N)
15: for each p ∈ S in decreasing HU_N(p) order do
16:    if RKS̄(N) has fewer than k products then
17:        Insert p into RKS̄(N)
18:    else
19:        T(N) ← the k-th largest LU_N value in RKS̄(N)
20:        if HU_N(p) > T(N) then
21:            Insert p into RKS̄(N)
22:        else
23:            Break                  ▷ RKS̄(N) is finalized; go to Line 24
24: if N is an internal node then
25:    for each child of node N do
26:        Insert(RKS̄(N), N.child)
```

Proof The expected cardinality of the k-skyband is $\frac{k \ln^{d-1} |\mathcal{P}|}{d!}$ [31]. Hence, computing the rk-skyband for a cell requires $O\left(\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2\right)$ r-dominance tests. Given (from Lemma 2) that the heat-map comprises $\binom{\lambda-1+d}{d} - \binom{\lambda}{d}$ cells, the stated complexity follows.

Lemma 5 *The space complexity of BL is* $O\left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)$.

Proof The space consumption of BL is dominated by the size of the heat-map. A corollary of Lemma 2 is that the heat-map comprises $\binom{\lambda-1+d}{d} - \binom{\lambda}{d}$ cells, resulting in the stated complexity.

The following lemma is essential for the analysis of CSA$^+$ and MDA$^+$.

Lemma 6 *The total number of nodes in the simplex pyramid is* $O\left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)$.

Proof Let ξ_i be the number of nodes in the i-th level of the pyramid, where $i \in [0, h]$ and $h = \log_2(\lambda)$ is the height of the pyramid. For any $d \geq 2$, each internal node is split by at least one axis-perpendicular hyper-plane, and thus it has at least two children. That is, $\xi_i \leq \frac{\xi_{i+1}}{2}, \forall i \in [0, h-1]$. The total number of nodes in the pyramid is:

$$\xi_h + \xi_{h-1} + \xi_{h-2} + \cdots + \xi_0 \leq \xi_h + \frac{\xi_h}{2^1} + \frac{\xi_h}{2^2} + \cdots + \frac{\xi_h}{2^h} < 2 \cdot \xi_h$$

Since $\xi_h = \binom{\lambda-1+d}{d} - \binom{\lambda}{d}$, the lemma follows.

Lemma 7 *The time complexity of CSA$^+$ is* $O\left(\frac{d-2}{d-1} \cdot \left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2 \cdot \left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)\right)$.

Proof Given that the expected cardinality of the k-skyband is $\frac{k \ln^{d-1} |\mathcal{P}|}{d!}$ [31], CSA$^+$ inserts into the simplex pyramid $O\left(\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2\right)$ product pairs.

The user spectrum is a $(d-1)$-dimensional object, partitioned in each pyramid level by axis-perpendicular hyper-planes. The produced hyper-plane arrangement has a dimensionality of $(d-1)$, and the cells of this arrangement correspond to nodes in the specific pyramid level. In a general $(d-1)$-dimensional arrangement, the zone theorem [24] states that out of all its $O(n^{d-1})$ cells, a hyper-plane cuts through only $O(n^{d-2})$ of them. Hence, we expect that a hyper-plane cuts through only $\frac{d-2}{d-1}$ of the nodes in a pyramid level. This means that, when inserting a product pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ into the pyramid, the hyper-plane $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_i) = \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}_j)$ cuts through only $\frac{d-2}{d-1}$ of the nodes in each level, i.e., r-dominance between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ is unclear (and thus the insertion routine needs to be recursively invoked) in $\frac{d-2}{d-1}$ of the nodes. It follows that, for each product pair inserted into the pyramid, r-dominance tests are performed for $O\left(\frac{d-2}{d-1} \cdot \left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)\right)$ nodes in total. Given that the number of inserted product pairs is $O\left(\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2\right)$, the stated complexity follows.

Lemma 8 *The space complexity of CSA$^+$ is* $O\left(\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2 \cdot \left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)\right)$.

Proof The space compexity of CSA$^+$ is determined by the (global) dominance graph Γ and the delta-graphs $\Delta(N)$ (one per node of the pyramid). Each of these graphs represents pairwise r-dominance for $O\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)$ products and, thus, takes $O\left(\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right)^2\right)$ space. Given that there are $O\left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)$ nodes in the pyramid, the lemma follows.

Lemma 9 *The time complexity of MDA$^+$ is* $O\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!} \cdot \log_2\left(\frac{k \ln^{d-1} |\mathcal{P}|}{d!}\right) \cdot \left(\binom{\lambda-1+d}{d} - \binom{\lambda}{d}\right)\right)$.

Proof Considering that $\overline{RKS}(N)$ is a tight superset of the rk-skyband for N, we expect the bottleneck in MDA$^+$ to be the computation of $\overline{RKS}(N)$ for each node in the simplex pyramid. The product set relayed to a node N includes $O(\frac{k \ln^{d-1} |\mathcal{P}|}{d!})$ products. The sorting of these products by $HU_N(\boldsymbol{p})$, and the use of a heap to maintain the

threshold $\mathcal{T}(N)$, take $O\left(\frac{k\ln^{d-1}|\mathcal{P}|}{d!}\cdot\log_2\left(\frac{k\ln^{d-1}|\mathcal{P}|}{d!}\right)\right)$ time each. Having $O\left(\binom{\lambda-1+d}{d}-\binom{\lambda}{d}\right)$ nodes in the pyramid, the stated complexity follows.

Lemma 10 *The space complexity of MDA$^+$ is* $O\left(\binom{\lambda-1+d}{d}-\binom{\lambda}{d}+\frac{k\ln^{d-1}|\mathcal{P}|}{d!}\log_2(\lambda)\right)$.

Proof The size of the simplex pyramid is proportional to the total number of its nodes, i.e., $O\left(\binom{\lambda-1+d}{d}-\binom{\lambda}{d}\right)$. At any point during execution, MDA$^+$ requires the $\overline{RKS}(N)$ set for $\log_2(\lambda)$ nodes that form a path from the pyramid's root down to a leaf. Given that each of these $\overline{RKS}(N)$ sets includes $O(\frac{k\ln^{d-1}|\mathcal{P}|}{d!})$ products, and adding the space occupied by the simplex tree, we derive the stated space complexity.

6 Discussion: Side-applications

In addition to the focal applications that motivate our study and are described in Introduction, our processing framework has interesting side-applications.

Identifying outstanding competitors: The utility-based (e.g., $MaxMin_k$) heat-map of $\mathcal{P}$ can be seen as a representation of the utility of the dataset's front-line in different parts of the user spectrum. Therefore, if some product $\boldsymbol{p}$ happens to achieve significantly higher utility than $MaxMin_k(c)$ in a cell c, this is a strong indication that $\boldsymbol{p}$ may be an exceptional product. This, in turn, enables applications that seek to identify outstanding competitors in the market. Observe that alternative aggregation measures could be used instead of $MaxMin_k$ in the heat-map, as well as alternative aggregations for $\boldsymbol{p}$'s utility over c (e.g., its average or minimum utility in c), but in all cases any product that stands out utility-wise will certainly be in $RKS(c)$. For the determination of whether a product's utility is sufficiently higher than the (aggregate) utility in $RKS(c)$ in order to be deemed outstanding, we could adapt any common outlier definition [57,54]. Alternatively, we could report the m products across the heat-map (for some $m \geq 1$) with the largest relative utility deviation from the respective $RKS(c)$.

Pre-computation for top-k query acceleration: The wide adoption of the top-k query in multi-objective settings has led to several methods for the pre-processing of a (static) product set in order to accelerate ad-hoc top-k queries. Early efforts exploit the convex hull properties [16], notions of "many-to-one dominance" [87], and the use of materialized ranked views [35,36,21]. A comprehensive coverage of these methods is provided in [29]. It turns out that the $MaxMin_k$ heat-map we produce offers a distinct and promising approach to this problem.

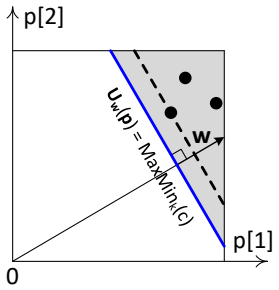


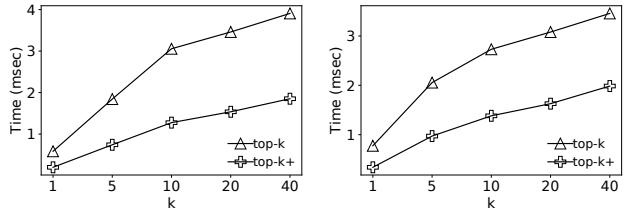
Fig. 5 Using the $MaxMin_k$ heat-map for top-k processing

Assume that, as pre-processing, we solve Problem 1 on $\mathcal{P}$ and materialize its $MaxMin_k$ heat-map. Let $\boldsymbol{w}$ be the weight vector of an ad-hoc top-k query, and $\mathcal{U}_{\boldsymbol{w}}^k$ be the (originally unknown) utility of the top-k-th product for $\boldsymbol{w}$. Using the heat-map as a look-up table, we can identify the cell c where $\boldsymbol{w}$ falls and retrieve $MaxMin_k(c)$. As explained in Section 3.1, $\mathcal{U}_{\boldsymbol{w}}^k$ is guaranteed to be no smaller than $MaxMin_k(c)$. This means that the top-k products for $\boldsymbol{w}$ must be among those that satisfy the condition $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}) \geq MaxMin_k(c)$, i.e., to products that lie above the hyperplane defined by $\sum_{i=1}^{d} p[i]w[i] = MaxMin_k(c)$ in the product space [75,60]. We may efficiently retrieve such products with a range query using a general purpose index on $\mathcal{P}$, and subsequently compute their utilities in order to determine the actual top-k among them.

Figure 5 presents an example in $d = 2$. Note that the illustrated domain is the product space. The line that corresponds to $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}) = MaxMin_k(c)$ is shown bold and solid, and it is perpendicular to $\boldsymbol{w}$. A range query on $\mathcal{P}$ for the shaded area is guaranteed to retrieve the top-k set ($k = 3$ is assumed). The dashed line corresponds to $\mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p}) = \mathcal{U}_{\boldsymbol{w}}^k$ (i.e., the actual utility of the top-k-th product for $\boldsymbol{w}$) and is guaranteed to be entirely within the shaded area.

Although this side-application is beyond the scope and application goals of this paper, we consider it a promising direction for future dedicated investigation; Figure 6 presents some preliminary, yet encouraging results. We plot the computation time of the sketched approach for 100 randomly generated top-k queries on two real datasets (properly introduced in Section 7) with $\lambda = 32$. Note that the reported measurements do not include the time to produce the heat-map, because in this setting it is assumed pre-computed. As a yardstick, we include the standard branch-and-bound top-k algorithm [73]. We observe a 2- to 3-fold improvement in processing time.

Utility heat-map with heterogeneous granularity: An interesting variant of Problem 1 is to enable a heterogeneous heat-map granularity across the user spectrum, depending on the dataset's characteristics. Intuitively, a large $RKS(c)$ means that there are many competitive products for c, and thus the aggregation of their utilities in a single value (by



(a) Running time (HOUSE) (b) Running time (NBA)

Fig. 6 Preliminary findings for top-k query acceleration

our competitiveness measure) may suppress their individual, more localized strengths. In that sense, if the cardinality of $RKS(c)$ is too large, it may be fairer (to the top products) and more informative (to the human analysts) to further partition c.

In particular, instead of specifying λ, we may input an integer m (which could be expressed, say, as a multiple of k) and impose the requirement that the rk-skyband for each leaf (i.e., cell) in the simplex pyramid includes no more than m products. In effect, the splitting of nodes will be dynamically decided (according to $|RKS(N)|$) and the resulting simplex pyramid will be an unbalanced tree. Concordantly, the size of the cells in the produced heat-map could vary greatly.

MDA$^+$ suits well this variant of the problem, since it anyway individualizes the $\overline{RKS}(N)$ computation per node. Specifically, after $\overline{RKS}(N)$ is finalized for a node, and before it is propagated to N's children (i.e., before Line 24 in Algorithm 2), if $|\overline{RKS}(N)|$ is no greater than m, the recursion to stop at N and N be treated as a leaf of the simplex pyramid, i.e., a cell of the heat-map. Given that $\overline{RKS}(N)$ is a superset of $RKS(N)$, the above process could unnecessarily split a node when $|\overline{RKS}(N)| > m \geq |RKS(N)|$, but that can be trivially resolved by merging the respective sibling cells (i.e., reversing the slit) in a post-processing step.

7 Empirical Investigations

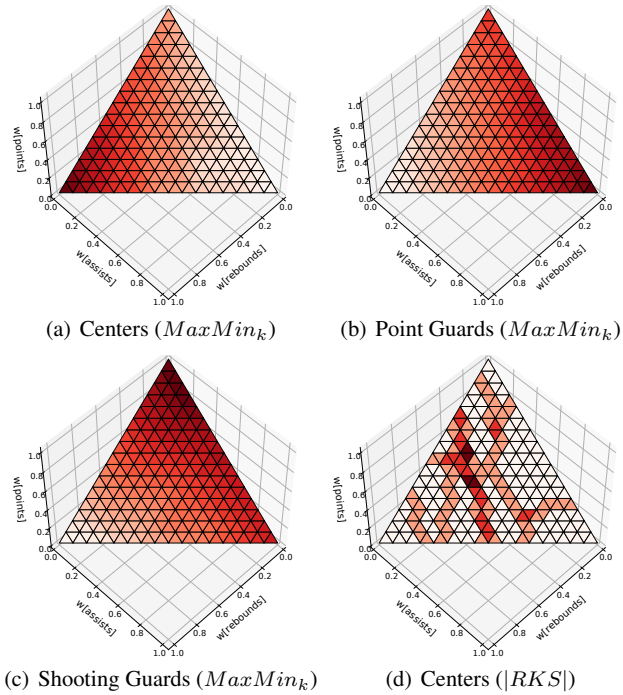
Our empirical investigations comprise two parts; case studies in Section 7.1, and performance experiments in Section 7.2.

7.1 Case Studies

Our competitiveness measures draw directly from the principles of multi-objective querying. To complement intuition, however, in our first case study we empirically validate their suitability. In particular, we use real datasets with known ground truth characteristics, and check whether our heat-maps can capture their pre-known strengths. Using NBA player statistics for season 2020-21 [4], we extracted 3 datasets corresponding to *Centers* (98 players), *Point Guards* (102), and *Shooting Guards* (123). In Figures 7(a), 7(b) and 7(c) we present their utility-based heat-maps for $k = 10$ and $\lambda = 16$, using rebounds, assists and points as the player attributes. **Fact:** *Centers are typically skilled at gathering rebounds; they may still score near the basket, but they rarely give assists* [1]. Our heat-map in Figure 7(a) indeed captures that fact, demonstrating a particular strength of the Center dataset in rebounds (left corner). Between their weaker aspects, i.e., assists and points, Centers are successfully drawn hotter in points (top corner). **Fact:** *Point Guards control the ball and make sure it gets to the right player at the right time; they lead their team in assists but, if open for a shot, they will take it to score a few points* [1]. The heat-map in Figure 7(b) accurately draws that picture for Point Guards, illustrating assists as their strongest ability (right corner), with point-scoring still hotter than rebounds. **Fact:** *Shooting Guards' main objective is to score points; moving around the 3-point line (far from the basket) they rarely gather rebounds, but may occasionally give assists* [1]. The heat-map in Figure 7(c) effectively captures these characteristics, with the top corner (points) being clearly the hottest, followed by the right one (assists). For completeness, in Figure 7(d) we present the competition-based ($|RKS|$) heat-map for Centers.[5] **Fact:** *Players in the same position (e.g., Centers) have the same performance objectives, thus their statistics are correlated around the position's main skill (rebounds)* [22]. When the user is looking for the stereotypical Center (i.e., for preferences close to the left corner in Figure 7(d)) there are few star Centers, who overshadow most others; i.e., minimal competition, and thus light coloring. When the user's preferences deviate from the norm for Centers (i.e., for preferences further from the left corner and towards the middle of the user spectrum) many mediocre Centers may appear competitive, because they do well in secondary aspects for the position, e.g., points and assists. The situation is similar/symmetric in the competition-based heat-maps of Point Guards and Shooting Guards (omitted for brevity).

In our second case study, we aim for useful insights in a real product and preference set, sourced from the TripAdvisor portal [5]. The data include ratings for 1,850 hotels on $d = 7$ aspects (namely, on value, room, location, cleanliness, front desk, service, and business service), comprising the product set $\mathcal{P}$. The TripAdvisor data also include text reviews and overall scores from actual users for these hotels. We employed the preference learning technique in [81] to mine the text and scores in the reviews to produce a 7-dimensional weight vector per user. The derived 137,563

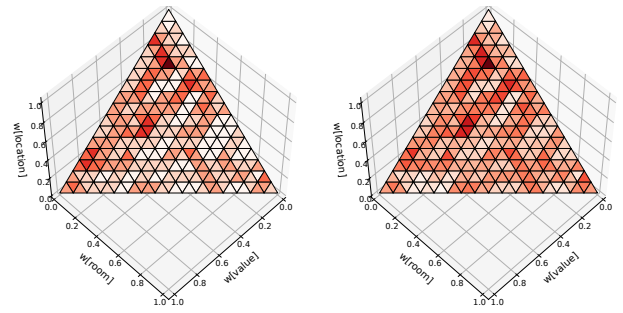
[5] The $\#DR$ heat-map is very similar to $|RKS|$, as we will elaborate shortly.



(a) Centers ($MaxMin_k$) (b) Point Guards ($MaxMin_k$)

(c) Shooting Guards ($MaxMin_k$) (d) Centers ($|RKS|$)

Fig. 7 Heat-maps for different playing positions in NBA



(a) $|RKS|$ competitiveness (b) $\#DR$ competitiveness

Fig. 8 $|RKS|$ and $\#DR$ heat-maps for TripAdvisor hotels

weight vectors serve as a representative sample of user demand.

We first focus on the hotel set $\mathcal{P}$ in isolation. For visualization, we initially use only the first $d = 3$ hotel attributes, i.e., ratings on value, room, and location. We set $k = 10$ and $\lambda = 16$, and present the utility-based (i.e., $MaxMin_k$) heat-map in Figure 1(a) (appearing in the Introduction) and the two competition-based (i.e., $|RKS|$ and $\#DR$) in Figure 8. There is a total of 256 cells, as can also be derived by Lemma 2 for $h = \log_2(\lambda) = 4$ and $d = 3$.

A first observation is that the heat-maps for $|RKS|$ and $\#DR$ look very similar. To quantify/verify this, we form a pair of values $(|RKS(c)|, \#DR(c))$ for each cell c in the heat-map, and compute the Pearson correlation coefficient[6] [56]; the coefficient is 0.934. This strong correlation is also evident in the full-dimensional heat-maps in $d = 7$ (where the coefficient is 0.852). To investigate the generality of this fact, Table 2 presents the Pearson correlation coefficient between $|RKS|$ and $\#DR$ for all benchmark datasets we use (and describe) in Section 7.2, for the default parameters in our performance evaluation. We use the real datasets (HOTEL, HOUSE, NBA) in their full dimensionality, and the synthetic ones (CORR, INDE, ANTI) in their default $d = 4$. Given the consistency of the correlation across all datasets, we henceforth use $\#DR$ as the representative competition-based measure.

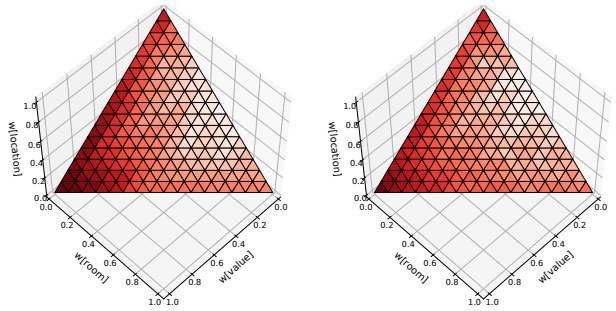
HOTEL	HOUSE	NBA	CORR	INDE	ANTI
0.955	0.882	0.856	0.854	0.964	0.986

Table 2 Pearson correlation coefficient between $|RKS|$ and $\#DR$

In contrast, when we compare Figures 1(a) and 8(b) we observe that utility-based competitiveness ($MaxMin_k$) is very different from competition-based ($\#DR$). Actually, the Pearson correlation coefficient is -0.164 in $d = 3$, and -0.144 in full dimensionality, indicating that the two measures are practically independent, a behavior that is consistent in all benchmark datasets (results omitted for brevity). This confirms that the two types of competitiveness assess different aspects of the market, with the former measuring the degree of user satisfaction in a cell, and the latter the intensity of competition among high-ranking products. Another interesting observation in this comparison is that the contrast (differential) in nearby cells is much higher in the competition-based heat-map, whereas transitions are much smoother in the utility-based one. The reason is that, in $MaxMin_k(c)$, the utility of products in $RKS(c)$ has a direct proportionality to the cell's coordinates. Instead, the r-dominance relations between products (which determine $|RKS|$ and $\#DR$) are not directly dictated by c's coordinates, but depend strongly also on the products' relative positions in the product space.

Looking into the semantics of Figure 1(a) and $MaxMin_k$, the deepest-colored cells indicate that the hotel market caters best for users who prioritize value over room quality and location. On the other hand, regarding the choice of target regions for a new hotel, the $MaxMin_k$ heat-map suggests that it may be easier/cheaper to introduce a high-ranking hotel when it is tailored to preferences in the lighter-colored cells; recall that $MaxMin_k(c)$ serves as a

[6] The Pearson correlation coefficient for n value pairs (x_i, y_i) is defined as $r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$, where $\bar{x}$ is the mean of x_i values, and analogously for $\bar{y}$. It takes values between -1 (perfect anti-correlation) and 1 (perfect correlation).



(a) $MaxMax_k$ competitiveness (b) $AvgMax$ competitiveness

Fig. 9 Other utility-based heat-maps for TripAdvisor hotels

lower bound for the utility a new product should have in order to enter the top-k set for any preferences in c.

Turning to Figure 8(b) and $\#DR$, we observe that (i) there are many light-colored (i.e., low competition) cells, and (ii) the light-colored cells are spread around the user spectrum. These facts suggest, respectively, that there is ample potential to set up a new hotel and, at the same time, that there is flexibility in choosing the target clientele (and thus in determining the intended niche) of the new hotel. Note that combining findings from both heat-maps (for $\#DR$ and for $MaxMin_k$) could provide a more holistic view on $\mathcal{P}$, and thus lead to a better choice of target regions (e.g., cells that are lightly-colored for both measures).

In the description of utility-based measures in Section 3.1, we explained that assigning a utility value to a cell c requires appropriate aggregation and that, although $MaxMin_k$ is intuitive and comes with a practical lower bound property, alternative aggregations are possible. As a side-investigation, in Figure 9 we plot utility-based heat-maps using two such alternatives. Specifically, in Figure 9(a) we use $MaxMax_k$; letting operator $\max^k(.)$ return the k-th largest from a set of values, $MaxMax_k(c) = \max^k_{\boldsymbol{p}\in RKS(c)} \max_{\boldsymbol{w}\in c} \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$. That is, instead of representing a product $\boldsymbol{p} \in RKS(c)$ by its minimum possible utility in c, we represent it by the maximum. By its definition, $MaxMax_k$ is an upper bound of the top-k-th utility in $\mathcal{P}$ for any $\boldsymbol{w} \in c$, and thus its heat-map is a more optimistic representation of the dataset's utility "fringe" compared to $MaxMin_k$. On the other hand, in Figure 9(b) we plot the $AvgMax$ heat-map, where $AvgMax(c) = \text{avg}_{\boldsymbol{p}\in RKS(c)} \max_{\boldsymbol{w}\in c} \mathcal{U}_{\boldsymbol{w}}(\boldsymbol{p})$. The $MaxMax_k$ and $AvgMax$ heat-maps are similar to Figure 1(a), as the measures are naturally correlated to $MaxMin_k$ (with Pearson correlation coefficients 0.945 and 0.951, respectively).

The final investigation in this case study brings in the picture the preference profiles ($d = 7$) extracted from TripAdvisor reviews. Each of our (full-dimensional, normalized) heat-maps can be seen as the probability mass function (PMF) that defines the distribution of the market's competitiveness (under the respective competitiveness measure). We may also obtain the PMF of the user demand by counting the number of weight vectors (out of the 137,563 we extracted) that fall in each cell. To assess the alignment between market competitiveness and user demand, we employ two established measures of distribution similarity, i.e., the Jensen–Shannon divergence [44] and the Bhattacharyya coefficient [14]. They both take values in $[0, 1]$, but absolute similarity corresponds to value 0 for the former, and 1 for the latter. They evaluate to 0.6955 and 0.3639 when comparing user demand with $MaxMin_k$, and to 0.6766 and 0.3815 when comparing user demand with $\#DR$. These values indicate that there is surely space and opportunities for new hotels and for further alignment of the market with the user demand.

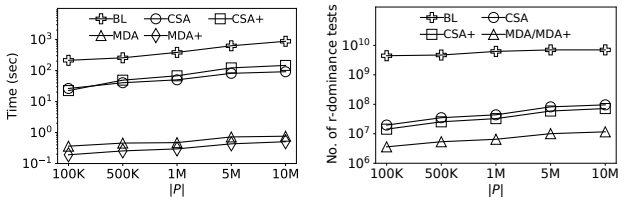
Parameter	**Tested and default values**
Product set cardinality $\|\mathcal{P}\|$	100K, **500K**, 1M, 5M, 10M
Dimensionality d	2, 3, **4**, 5, 6, 7
Parameter k	1, 5, **10**, 20, 40
Granularity parameter λ	8, 16, **32**, 64, 128
Product dataset	CORR, **INDE**, ANTI, HOTEL, HOUSE, NBA

Table 3 Tested values and defaults of problem parameters

7.2 Performance Experiments

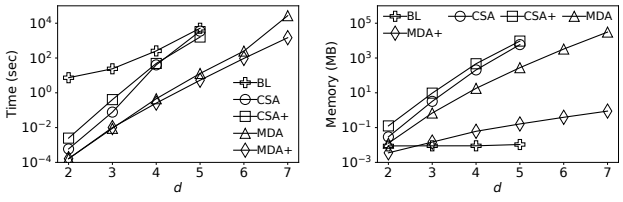
In this section, we evaluate performance. As product set $\mathcal{P}$, we use standard synthetic datasets, i.e., correlated (CORR), independent (INDE), and anti-correlated (ANTI), that represent typical product distributions in multi-objective decisions [15], as well as real datasets HOTEL, HOUSE, NBA, which are common benchmarks in the literature. HOTEL includes $d = 4$ attributes for 418,843 hotels [2]; HOUSE $d = 6$ types of expenditure for 315,265 households [3]; and NBA $d = 8$ statistics for 21,960 NBA players [4]. Table 3 summarizes the tested ranges and default values (in bold) of the problem parameters. In each experiment we vary one parameter and keep the remaining ones to their defaults. The performance of all methods varies insignificantly with the competitiveness measure used, as it is heavily dominated by the derivation of $RKS(c)$ for each cell in the heat-map; we use $MaxMin_k$ by default. All algorithms were implemented in C++, and complied by GCC with Level 2 optimization. We used a computer with Intel Xeon Gold 5122, 3.6GHz CPU and 32GB RAM, running on CentOS.

In Figure 10, we present the running time and the number of r-dominance tests for different product set cardinalities. The response time of all methods increases sub-linearly to $|\mathcal{P}|$, which is reasonable since the k-skyband and, similarly, the rk-skyband generally grow in size in the same



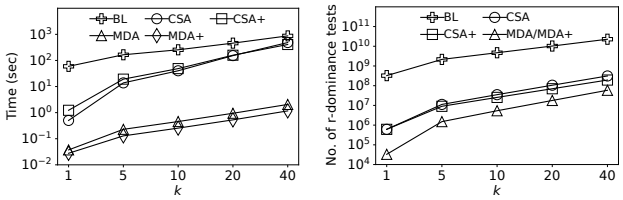
(a) Running time (b) No. of r-dominance tests

Fig. 10 Effect of $|\mathcal{P}|$



(a) Running time (b) Memory consumption

Fig. 11 Effect of d



(a) Running time (b) No. of r-dominance tests

Fig. 12 Effect of k

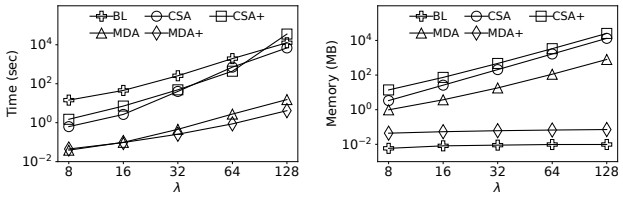
(sub-linear) manner [31]. **First**, comparing MDA/MDA$^+$ to CSA/CSA$^+$, there is a running time improvement of 2 orders of magnitude. This behavior is as expected, due to CSA/CSA$^+$ making one insertion into the simplex pyramid for *each pair* of products in the k-skyband (as opposed to only one insertion per k-skyband product in MDA/MDA$^+$) and the fact that they practically compute the rk-skyband even for the internal nodes of the pyramid. That is also reflected in the number of r-dominance tests in Figure 10(b), where MDA/MDA$^+$ perform 5.3M tests in the default setting, versus 25M and 35M tests for CSA$^+$ and CSA, respectively. **Second**, between MDA and MDA$^+$, the latter is the clear winner, being 2 times faster. By design, MDA and MDA$^+$ perform an identical number of r-dominance tests. Hence, the performance difference between them is purely due to the $\overline{RKS}(N)$ computation in the internal nodes of the pyramid. This testifies to the effectiveness of (i) MDA$^+$'s individualization of the $\overline{RKS}(N)$ computation per node, and (ii) its avoidance of unnecessary product propagations in the sub-trees of considered nodes. MDA$^+$ exhibits strong scalability, requiring 0.5s even for the largest product set tried ($|\mathcal{P}| = 10M$). **Third**, the comparison between CSA and CSA$^+$ has no clear winner. As shown in Figure 10(b), CSA$^+$ indeed performs fewer r-dominance tests than CSA. However, the saved computations are often overturned by CSA$^+$'s overhead to maintain the delta-graphs in the pyramid. **Fourth**, BL is the slowest method, being 3 orders of magnitude slower than MDA/MDA$^+$and 1 order slower than CSA/CSA$^+$. This, and its excessive number of r-dominance tests (in Figure 10(b)) confirm the BL deficiencies we elaborated at the end of Section 4.1; in the default setting (i.e., $|\mathcal{P}| = 500K$) it performs 4.7 *billion* tests!

In Figure 11, we vary the dimensionality d and measure the running time and memory consumption. From Lemma 2, we see that the number of cells in the heat-map increases near-exponentially with d. This explains the increase in time and space requirements for all algorithms. Note that for $d > 5$, BL fails to terminate within a day; also, CSA and CSA$^+$ run out of memory. This highlights that the advantages of MDA$^+$ do not stop at processing time, but they extend to space overhead too; it is indicative that even for $d = 7$, MDA$^+$ requires only about 1MB of space. Recall that CSA and CSA$^+$ maintain a rk-skyband for each node in the pyramid, in the form of a dominance count array and a (even more voluminous) delta-graph, respectively. MDA relies on a lighter structure, i.e., $\overline{RKS}(N)$, yet it still needs to keep a $\overline{RKS}(N)$ set for each and every node in the pyramid. MDA$^+$, on the other hand, does not require many intermediate results, since it completes the $\overline{RKS}(N)$ computation before propagating it to N's children. Actually, as explained at the end of Section 4.3.2, MDA$^+$ needs to maintain the $\overline{RKS}$ set for a maximum of $h = \log_2(\lambda)$ nodes at any point during execution. Furthermore, while CSA, CSA$^+$ and MDA require an explicit representation of the entire simplex pyramid, MDA$^+$ requires the geometric representation of only h nodes.[7] Regarding BL, the space requirements are minimal (below 20KB) and only slightly affected by d, since its $RKS(c)$ computations are independent from each other; its requirements come predominantly from the search heap of its rk-skyband computation building block [50].

Figure 12 studies the effect of k. Regarding BL, a larger k implies a larger rk-skyband for every cell in the heat-map. Similarly, it leads to larger rk-skybands and $\overline{RKS}(N)$ sets (for CSA/CSA$^+$ and MDA/MDA$^+$, respectively) throughout the nodes of the simplex pyramid. This results in more r-dominance tests and, thus, in increased running time. That said, not all methods are affected to the same extent, with MDA and MDA$^+$ scaling significantly better.

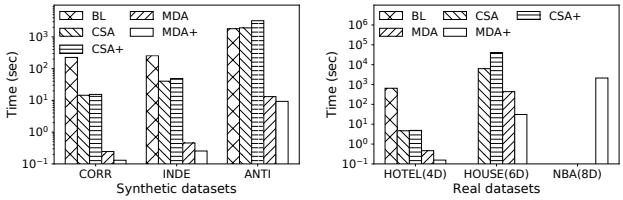
In Figure 13, we vary the heat-map granularity. For λ equal to 8, 32, and 128 we have a total of 260, 16,400, and 1,048,640 cells, respectively. Hence, the time and space re-

[7] Note that the space measurements do not include the size of the heat-map itself, since the per-cell competitiveness values are directly output upon $RKS(c)$ computation.



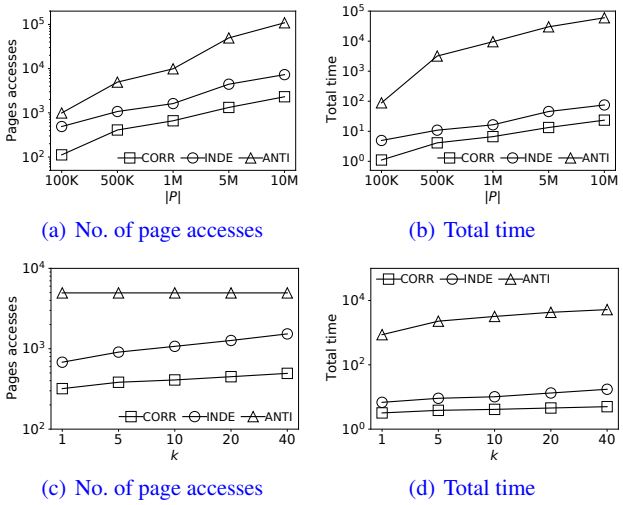
(a) Running time (b) Memory consumption

Fig. 13 Effect of λ



(a) Synthetic distributions (b) Real product sets

Fig. 14 Synthetic distributions and real product sets



(a) No. of page accesses (b) Total time

(c) No. of page accesses (d) Total time

Fig. 15 Secondary storage scenario

quirements increase for all methods. However, the increase is sub-linear to the number of cells. That is because as the number of cells grows, their volume decreases. In turn, this means that the rk-skyband for each cell includes fewer products and, similarly, that we need to consider fewer products for its computation. Focusing on running time (Figure 13(a)), the difference of MDA$^+$ from MDA grows with λ, because a higher simplex pyramid implies greater gains from the avoidance of unnecessary propagations. Another observation is that the running time of CSA$^+$ increases more sharply than the other methods, because the number of nodes in the pyramid, and thus the number of delta-graphs it needs to maintain, shoots up.

In Figure 14(a), we use the synthetic product sets. The fastest (slowest) distribution to process is CORR (ANTI, respectively). That is because in CORR the rk-skyband generally holds the fewest products, while in ANTI the most. Our best method, MDA$^+$ remains practical in all settings, requiring just 9.3s even for ANTI.

In Figure 14(b), we use real datasets. The results are in line with previous figures, and especially with Figure 11, as they demonstrate the strong effect of d; NBA, although smaller than HOTEL and HOUSE, is the slowest to process due to its higher dimensionality. Note that BL fails to terminate within a day for HOUSE and NBA. On the other hand, CSA, CSA$^+$, and MDA run out of memory for NBA, making MDA$^+$ the only feasible method for it (with space requirements of only 44KB).

8 Conclusion

Identifying the products of most interest to a user has been well-explored in the database literature, giving rise to standard operators such as the skyline and the top-k query. In this paper, we take a different viewpoint of this general querying model, and aim instead to chart the competitiveness of a multi-attribute dataset at different parts of the users' preference spectrum. We identify two orthogonal types of market competitiveness, and define measures for them. We design a suite of algorithms for the efficient computation of these measures across the user spectrum, and represent them in the form of a heat-map. We perform case studies with real data that reveal actionable market insights, and we demonstrate experimentally the efficiency and scalability of our algorithms. Byproducts of our work include the identification of products that stand out in the competitive landscape, and a new approach for top-k query acceleration.

Both classes of our competitiveness measures are defined on the restricted k-skyband (i.e., the set of products that matter preference-wise) for different parts of the user spectrum. Therefore, we have focused our technical attempts to explicitly compute that set for every cell of the heat-map. A challenging direction for future work would be to evaluate (or approximate) the competitiveness measures without explicitly computing the restricted k-skyband, in order to further reduce the computation time. In the case of approximation, the challenge entails also the formal bounding of the inaccuracy incurred.

References

1. Basketball positions. https://en.wikipedia.org/wiki/Basketball_positions.

2. Hotel dataset. https://github.com/RyanBalfanz/hotelsbase.
3. House dataset. https://www.ipums.org.
4. NBA dataset. https://www.basketball-reference.com.
5. TripAdvisor dataset. https://www.cs.virginia.edu/~hw5x/dataset.html.
6. E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD Conference*, pages 515–526, 2006.
7. E. Achtert, H. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*, volume 360, pages 886–897, 2009.
8. P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Efficient algorithms for k-regret minimizing sets. In *SEA*, volume 75 of *LIPIcs*, pages 7:1–7:23, 2017.
9. A. Asudeh, A. Nazi, N. Zhang, and G. Das. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *SIGMOD Conference*, pages 821–834, 2017.
10. A. Asudeh, A. Nazi, N. Zhang, G. Das, and H. V. Jagadish. RRR: rank-regret representative. In *SIGMOD Conference*, pages 263–280, 2019.
11. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, pages 322–331, 1990.
12. M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.
13. T. Bernecker, T. Emrich, H. Kriegel, M. Renz, S. Zankl, and A. Züfle. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. *PVLDB*, 4(10):669–680, 2011.
14. A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.
15. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
16. Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *SIGMOD Conference*, pages 391–402, 2000.
17. S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *PVLDB*, 7(5):389–400, 2014.
18. J. Church and R. Ware. *Industrial Organization: A Strategic Approach*. Irwin McGraw Hill, 2000.
19. P. Ciaccia and D. Martinenghi. Reconciling skyline and ranking queries. *PVLDB*, 10(11):1454–1465, 2017.
20. P. Ciaccia and D. Martinenghi. FA + TA $<$ fsa: Flexible score aggregation. In *CIKM*, pages 57–66, 2018.
21. G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *VLDB*, pages 451–462, 2006.
22. J. Duch, J. S. Waitzman, and L. A. N. Amaral. Quantifying the performance of individual players in a team activity. *PLOS ONE*, 5(6):1–7, 06 2010.
23. J. S. Dyer and R. K. Sarin. Measurable multiattribute value functions. *Oper. Res.*, 27(4):810–822, 1979.
24. H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993.
25. T. Emrich, H. Kriegel, P. Kröger, M. Renz, and A. Züfle. Boosting spatial pruning: on optimal pruning of mbrs. In *SIGMOD Conference*, pages 39–50, 2010.
26. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
27. W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 1968.
28. J. Fürnkranz and E. Hüllermeier, editors. *Preference Learning*. Springer, 2010.
29. S. Ge, L. H. U, N. Mamoulis, and D. W. Cheung. Efficient all top-k computation - A unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Trans. Knowl. Data Eng.*, 25(5):1015–1027, 2013.
30. K. Gerardi and A. Shapiro. Does competition reduce price dispersion? new evidence from the airline industry. *Journal of Political Economy*, 117(1):1–37, 2009.
31. P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.
32. X. Han, B. Wang, J. Li, and H. Gao. Ranking the big sky: efficient top-k skyline computation on massive data. *Knowl. Inf. Syst.*, 60(1):415–446, 2019.
33. J. Hausman, G. Leonard, and J. D. Zona. Competitive analysis with differentiated products. *Annals of Economics and Statistics*, 34:159–180, 1994.
34. K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB J.*, 21(3):359–384, 2012.
35. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD Conference*, pages 259–270, 2001.
36. V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.*, 13(1):49–70, 2004.
37. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comp. Surveys*, 40(4):11:1–11:58, 2008.
38. C. Kalyvas and T. Tzouramanis. A survey of skyline query processing. *CoRR*, abs/1704.01788, 2017.
39. R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Wiley, 1976.
40. M. R. Kolahdouzan and C. Shahabi. Voronoi-based K nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.
41. F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD Conference*, pages 201–212, 2000.
42. H. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *ICDE*, pages 261–272, 2010.
43. C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. DADA: a data cube for dominant relationship analysis. In *SIGMOD Conference*, pages 659–670, 2006.
44. J. Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Inf. Theory*, 37(1):145–151, 1991.
45. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.
46. J. Liu, L. Xiong, Q. Zhang, J. Pei, and J. Luo. Eclipse: Generalizing kNN and skyline. In *ICDE*, pages 972–983. IEEE, 2021.
47. M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*, pages 356–365, 2008.
48. K. Mouratidis, K. Li, and B. Tang. Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size. In *SIGMOD Conference*, pages 1317–1330, 2021.
49. K. Mouratidis, Y. Lin, and M. L. Yiu. Preference queries in large multi-cost transportation networks. In *ICDE*, pages 533–544, 2010.
50. K. Mouratidis and B. Tang. Exact processing of uncertain top-k queries in multi-criteria settings. *PVLDB*, 11(8):866–879, 2018.
51. K. Mouratidis, J. Zhang, and H. Pang. Maximum rank query. *PVLDB*, 8(12):1554–1565, 2015.
52. K. Mulmuley. On levels in arrangements and voronoi diagrams. *Discrete & Computational Geometry*, 6:307–338, 1991.
53. D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1):1114–1124, 2010.

54. National Institute of Standards and Technology. NIST/SEMATECH e-handbook of statistical methods; 7.1.6. what are outliers in the data, 2013.
55. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
56. K. Pearson. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London Series I*, 58:240–242, 1895.
57. R. K. Pearson. *Univariate Outlier Detection*, chapter 3, pages 69–91. 2005.
58. P. Peng and R. C. Wong. k-hit query: Top-k query with probabilistic utility function. In *SIGMOD Conference*, pages 577–592, 2015.
59. L. Qian, J. Gao, and H. V. Jagadish. Learning user preferences by adaptive pairwise comparison. *PVLDB*, 8(11):1322–1333, 2015.
60. S. Rahul and Y. Tao. Efficient top-k indexing via general reductions. In *PODS*, pages 277–288. ACM, 2016.
61. A. D. Sarma, A. Lall, D. Nanongkai, and J. J. Xu. Randomized multi-pass streaming skyline algorithms. *PVLDB*, 2(1):85–96, 2009.
62. M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, pages 751–762, 2006.
63. M. Sharifzadeh and C. Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *PVLDB*, 3(1):1231–1242, 2010.
64. M. Sharifzadeh, C. Shahabi, and L. Kazemi. Processing spatial skyline queries in both vector spaces and spatial network databases. *ACM Trans. Database Syst.*, 34(3):14:1–14:45, 2009.
65. C. Sheng and Y. Tao. Worst-case i/o-efficient skyline algorithms. *ACM Trans. Database Syst.*, 37(4):26:1–26:22, 2012.
66. M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with uncertain scoring functions: semantics and sensitivity measures. In *SIGMOD Conference*, pages 805–816, 2011.
67. I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse nearest neighbor queries for dynamic databases. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 44–53, 2000.
68. Y. Sun, R. Zhang, A. Y. Xue, J. Qi, and X. Du. Reverse nearest neighbor heat maps: A tool for influence exploration. In *ICDE*, pages 966–977, 2016.
69. B. Tang, K. Mouratidis, and M. Han. On m-impact regions and standing top-k influence problems. In *SIGMOD Conference*, pages 1784–1796, 2021.
70. B. Tang, K. Mouratidis, and M. L. Yiu. Determining the impact regions of competing options in preference space. In *SIGMOD Conference*, pages 805–820, 2017.
71. B. Tang, K. Mouratidis, M. L. Yiu, and Z. Chen. Creating top ranking options in the continuous option and preference space. *PVLDB*, 12(10):1181–1194, 2019.
72. Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.
73. Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3):424–445, 2007.
74. Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse k NN search. *VLDB J.*, 16(3):293–316, 2007.
75. P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, pages 277–288, 2003.
76. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.
77. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Monochromatic and bichromatic reverse top-k queries. *IEEE Trans. Knowl. Data Eng.*, 23(8):1215–1229, 2011.
78. A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1):364–372, 2010.
79. Q. Wan, R. C. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *PVLDB*, 2(1):898–909, 2009.
80. Q. Wan, R. C. Wong, and Y. Peng. Finding top-k profitable products. In *ICDE*, pages 1055–1066, 2011.
81. H. Wang, Y. Lu, and C. Zhai. Latent aspect rating analysis on review text data: a rating regression approach. In *KDD*, pages 783–792, 2010.
82. R. C. Wong, M. T. Özsu, P. S. Yu, A. W. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.
83. Y. Wu, J. Gao, P. K. Agarwal, and J. Yang. Finding diverse, high-value representatives on a surface of answers. *PVLDB*, 10(7):793–804, 2017.
84. T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *ICDE*, pages 1397–1399, 2008.
85. M. Xie, R. C. Wong, and A. Lall. An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. *VLDB J.*, 29(1):147–175, 2020.
86. M. Xie, R. C. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *SIGMOD Conference*, pages 959–974, 2018.
87. D. Xin, C. Chen, and J. Han. Towards robust indexing for ranked queries. In *VLDB*, pages 235–246, 2006.
88. G. Yang and Y. Cai. Querying improvement strategies. In *EDBT*, pages 294–305, 2017.
89. J. Yang, Y. Zhang, W. Zhang, and X. Lin. Influence based cost optimization on user preference. In *ICDE*, pages 709–720, 2016.
90. S. Yang, M. A. Cheema, X. Lin, and Y. Zhang. SLICE: reviving regions-based pruning for reverse k nearest neighbors queries. In *ICDE*, pages 760–771, 2014.
91. M. L. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *VLDB J.*, 18(3):695–718, 2009.
92. A. Yu, P. K. Agarwal, and J. Yang. Top-k preferences in high dimensions. *IEEE Trans. Knowl. Data Eng.*, 28(2):311–325, 2016.
93. P. Zhang, R. Cheng, N. Mamoulis, M. Renz, A. Züfle, Y. Tang, and T. Emrich. Voronoi-based nearest neighbor search for multi-dimensional uncertain databases. In *ICDE*, pages 158–169, 2013.
94. Z. Zhang, C. Jin, and Q. Kang. Reverse k-ranks query. *PVLDB*, 7(10):785–796, 2014.