Exact Processing of Uncertain Top-k Queries in Multi-criteria Settings

Kyriakos Mouratidis
School of Information Systems
Singapore Management University
kyriakos@smu.edu.sg

Bo Tang
Shenzhen Key Lab of Computational Intelligence
Southern University of Science and Technology
tangb3@sustc.edu.cn

ABSTRACT

Traditional rank-aware processing assumes a dataset that contains available options to cover a specific need (e.g., restaurants, hotels, etc) and users who browse that dataset via top-k queries with linear scoring functions, i.e., by ranking the options according to the weighted sum of their attributes, for a set of given weights. In practice, however, user preferences (weights) may only be estimated with bounded accuracy, or may be inherently uncertain due to the inability of a human user to specify exact weight values with absolute accuracy. Motivated by this, we introduce the *uncertain top-k query* (UTK). Given uncertain preferences, that is, an approximate description of the weight values, the UTK query reports all options that may belong to the top-k set. A second version of the problem additionally reports the exact top-k set for each of the possible weight settings. We develop a scalable processing framework for both UTK versions, and demonstrate its efficiency using standard benchmark datasets.

PVLDB Reference Format:
Kyriakos Mouratidis and Bo Tang. Exact Processing of Uncertain Top-k Queries in Multi-criteria Settings. *PVLDB*, 11 (8): 866-879, 2018.
DOI: https://doi.org/10.14778/3204028.3204031

1. INTRODUCTION

The top-k query has been studied extensively, and is considered the norm for multi-criteria decision making [25]. The traditional top-k query receives as input a dataset with d-dimensional records (options), and a vector $\boldsymbol{w}$ of d weights that specify the relative significance of each dimension (data attribute) for the user. The score of a record is defined as the weighted sum of its attribute values. The k highest-scoring records form the output of the top-k query. The weight vector $\boldsymbol{w}$ represents the user's preferences, and thus is key in producing useful recommendations. Usually, it is assumed to be input directly by the user. Alternatively, it may be mined from the user's past choices/behavior [28, 9, 27, 33], estimated by having the user make pairwise comparisons for a number of testing pairs of alternatives [26, 41], etc.

Our work is motivated by the crucial role of weight accuracy in the practicality and usefulness of the top-k query. We argue that, although convenient in terms of query processing, the assumption/requirement that the exact weight vector is known is hardly realistic. Be it specified directly by the user or by a preference learning algorithm, it is inherently inaccurate.

Consider a user who manually indicates weights for attributes *Service*, *Cleanliness*, and *Location*, when looking for a hotel on a hospitality portal. Based on pure intuition, she specifies weights 0.3, 0.5, and 0.2, respectively. While this setting makes top-k processing easy, it is an unfair and impractical request from the user to specify weights with absolute accuracy (and bear the consequences of a poor recommendation) when a minor change in a weight (e.g., from 0.3 to 0.29) could significantly alter the top-k set. Instead, it would be wiser if user inputs were dealt with as mere indications, and a leeway were given for inaccuracies in weight setting. Similarly, in preference learning techniques, the computed weight vector should only be used as a rough guide, rather than taken as an exact representation of user preferences. A natural way to deal with this issue, and offer a more user-centric and practical design, is to expand the (explicitly input or computationally learned) weight vector into a region[1], and report all possible top-k sets to the user. This task is accomplished by the *uncertain top-k query* (UTK).

The usability of UTK extends further. After reviewing the result of a regular top-k query, the user may wish to explore other similar options. E.g., on our hospitality portal, the user may ask for additional hotel recommendations, on top of the top-k already received. To offer alternatives of possible interest, we may expand the weight vector into a region, and report the additional options identified by UTK. Practically, that is equivalent to providing the user with top-k results for similar preference profiles.

Consider Figure 1, where each hotel's attributes are its average guest ratings on *Service*, *Cleanliness*, and *Location*, on a scale of 0 to 10. On the right, we illustrate the *preference domain*, i.e., the domain of the weight vector[2]. The axes correspond to w_1 and w_2, i.e., to the weights for *Service* and *Cleanliness*, on a scale of 0 to 1. The UTK input includes value k and the region of interest R. The latter is a

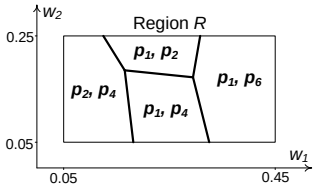
Proceedings of the VLDB Endowment, Vol. 11, No. 8

DOI: https://doi.org/10.14778/3204028.3204031

[1]Several preference learning techniques, like [41], already produce such a region, instead of a specific weight vector.
[2]As we will explain in Section 3.1, for 3-dimensional data, the preference domain is practically 2-dimensional.

Hotel	Svc.	Cln.	Loc.
p_1	8.3	9.1	7.2
p_2	2.4	9.6	8.6
p_3	5.4	1.6	4.1
p_4	2.6	6.9	9.4
p_5	7.3	3.1	2.4
p_6	7.9	6.4	6.6
p_7	8.6	7.1	4.3

(a) Data records



(b) Pref. domain and UTK result

Figure 1: Test data and UTK result for $k = 2$

convex polygon in the preference domain, representing the approximated or expanded user preferences. Here, $k = 2$ and R is the axis-parallel rectangle $[0.05, 0.45] \times [0.05, 0.25]$.

We distinguish two UTK versions. The first reports all the hotels that may belong to the top-2 set if $\boldsymbol{w}$ is free to lie anywhere in R. Importantly, the result must be *minimal*, i.e., for every hotel $\boldsymbol{p}$ in the UTK output, there should be at least one weight vector in R for which $\boldsymbol{p}$ is in the top-2 set. On our test data, the output is $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_4, \boldsymbol{p}_6\}$. The second UTK version provides more detail; it reports the specific top-2 set for any possible positioning of $\boldsymbol{w}$ in R. Its output in our example is the partitioning of R shown on the right, where each partition is associated with the top-2 set when $\boldsymbol{w}$ falls in it. E.g., for any weight vector in the leftmost partition in R, the top-2 hotels are $\{\boldsymbol{p}_2, \boldsymbol{p}_4\}$.

Previous research on top-k processing has only considered uncertainty in the data (e.g., [43, 6]), but not in the weight vector. The data uncertainty typically assumed is at the record or at the attribute level [19]. In the former case, each record may exist in the dataset with a certain probability. In the latter, the score of every record is a random variable that follows a given distribution. In both cases, there is no uncertainty in the weight vector, and the top-k answer is probabilistic (e.g., the top-3 set is $\{\boldsymbol{p}_2, \boldsymbol{p}_4, \boldsymbol{p}_5\}$ with probability 43%). Also, the records' scores are independent. In contrast, there is nothing probabilistic in UTK (e.g., the output is exact) and, importantly, the records' scores are necessarily correlated (since they all depend on $\boldsymbol{w}$), and vary together as $\boldsymbol{w}$ is freely positioned inside R.

Our contribution is twofold. First, we define a novel problem of practical significance, as it enables a more flexible way to express user preferences in top-k queries, by taking into account inherent inaccuracies in weight specification. Second, we propose a suite of comprehensive algorithms for it. Using standard benchmarks, we demonstrate that (i) our techniques offer practical response times for voluminous datasets, and (ii) they outperform by one to two orders of magnitude baselines constructed from previous work.

2. RELATED WORK

In the last decade there have been several studies on top-k queries in uncertain databases. Uncertainty in these studies pertains to the data, not to the weight vector. In a type of considered uncertainty, each record is associated with an existence probability, i.e., it may or may not be valid [43, 49, 24, 21, 31]. In the multitude of possible worlds implied (i.e., possible dataset versions), top-k semantics are unclear, giving rise to several formulations, e.g., to compute the most probable top-k set, the record most likely to rank i-th, the records that have a probability greater than a cutoff to be among the top-k, etc. In another type of uncertainty, possibly coexistent with the previous type, the score of each record is a (discrete or continuous) random variable, with a known probability distribution [6, 19, 42, 30]. E.g., a record's score could be 1.2 with probability 30% or 1.8 with probability 70%, or it could follow a continuous probability distribution. All these methods deal with uncertain data, rather than uncertain queries. This is crucial, since record scores are independent from each other, while in UTK the scores of all records are correlated. Also, UTK has no probabilities associated with, or affecting, its output.

Soliman et al. [44], on the assumption that the weight vector is equally likely to be anywhere in the preference domain, compute the most likely top-k result, as well as the ranking of the dataset that is the most representative of all possible score-wise rankings. Note that they report a single top-k result or a single ordering of the dataset, which is fundamentally different from UTK. They also define two sensitivity measures for top-k queries. On result sensitivity, for a given $\boldsymbol{w}$, Zhang et al. [52] compute the maximal region around $\boldsymbol{w}$ where the top-k result remains the same.

Specified the probability distribution of $\boldsymbol{w}$, Peng and Wong [40] identify a subset of m records from the dataset, so that the top-1 record for a random $\boldsymbol{w}$ has the largest probability to be in the subset. Related is also the computation of k-regret sets [37, 39, 17, 12], where a subset of m records is chosen, such that the top-1 record in that subset does not score much lower than the k-th record in the entire dataset for any weight vector. Methods on this topic report a requested number of records as representatives of the dataset, in contrast to UTK, which reports all records that may belong to the top-k set (or all possible top-k sets) when $\boldsymbol{w}$ is in R.

Ciaccia and Martinenghi [18] consider weighted L_p norms, i.e., a family of scoring functions that is a superset of linear ones. Given a set of linear conditions on the weights, they report all records in a dataset that could rank as the top-1 for any permissible weight combination. Their work is tailored strictly to $k = 1$. For linear scoring functions, their problem is equivalent to computing a part of the convex hull, a standard notion discussed later in this section.

Vlachou et al. [48] study reverse top-k queries, among which the monochromatic version is related to UTK. It determines all regions in the preference domain where a record $\boldsymbol{p}$ ranks among the top-k. [48] provides an algorithm specifically for 2-dimensional data. Tang et al. [45] propose the $kSPR$ methodology to answer that query for $d \geq 2$. Each competitor of $\boldsymbol{p}$ is mapped into a half-space in preference domain, where it scores higher than $\boldsymbol{p}$. In the resulting half-space arrangement, the cells that are inside fewer than k half-spaces form the output. $kSPR$ employs several types of pruning to reduce the number of considered competitors. Although designed for a different problem, we use $kSPR$ as a building block for a baseline UTK approach.

Mouratidis et al. [35] compute the highest rank that a given record could attain in a dataset of alternatives, for any weight vector. They also identify the regions of the preference domain which correspond to this rank. The problem is intrinsically different from UTK, and the techniques inapplicable to our case.

Application-wise, UTK is related to the *skyline*, and more so to its generalization, the *k-skyband*. A record *dominates* another if it has no smaller value in any dimension, and the records do not coincide [11]. The skyline of a dataset comprises those records that are not dominated by any other.

The k-skyband contains those records that are dominated by fewer than k others [38]. The k-skyband is proven to be a superset of all records that could appear in the top-k result for any weight vector. The standard algorithm for k-skyband computation, called *BBS* [38], utilizes a spatial index on the dataset, and follows the *branch-and-bound* paradigm [29]. Using a min-heap, it visits index nodes and data records in increasing distance from the *top corner* of the data domain (i.e., the point with maximum value in all dimensions); each index node is represented by the top corner of its minimum bounding box (MBB). *BBS* maintains an (initially empty) k-skyband set. When an index node is popped from the heap, if its top corner is dominated by fewer than k records in the k-skyband set, its children are pushed into the heap. When a record $\boldsymbol{p}$ is popped, if dominated by fewer than k records in the k-skyband set, it is inserted into the set. Instead of distance to top corner, *BBS* can use any monotone metric to determine the popping order; correctness is ensured as long as the metric guarantees that any record popped after $\boldsymbol{p}$ cannot dominate $\boldsymbol{p}$.

In group-based skyline formulations, each skyline member is a group of k records. E.g., in [32] a group dominates another if there are permutations of the two groups such that for every position $i \in [1, k]$ the i-th record in the first group either dominates or coincides with the i-th record in the second (and in at least one position the records do not coincide). Observe that no preferences are input by the user, unlike UTK where the output depends on the (approximate) preferences specified via region R. E.g, in Figure 1, $\boldsymbol{p}_7$ is not dominated by any other hotel, and will thus appear in at least one skyline group reported by [32]. In contrast, $\boldsymbol{p}_7$ does not appear in the UTK result, because it cannot make it in the top-2 hotels anywhere in R. This also demonstrates an additional fundamental difference. Dominance between two groups boils down to a number of per-record (i.e., standard) dominance tests. In UTK dominance tests do not suffice, e.g., two or more records that do not dominate $\boldsymbol{p}$, may still collaboratively disqualify $\boldsymbol{p}$ if they score higher than it at different parts of R, collectively preventing it from entering the top-k set at any position in R.

Hidden or unknown preferences have been considered in the context of skylines. E.g., specified a set of desirable and a set of undesirable records by the user, Mindolin and Chomicki [34] determine whether some criteria (dimensions) are more important than others. I.e., based on user examples on what should and what should not be in the skyline, they extract the underlying importance relations among criteria. Conceptually, that is inverse to UTK, where user preferences are explicitly given (by means of R), and preferred records (in top-k sets) are unknown/to be computed.

There is a relation between UTK and the *convex hull*. The convex hull is the smallest convex polytope that encloses all records in a dataset [8]. The top-1 record for any weight vector $\boldsymbol{w}$ is guaranteed to be on the convex hull. Chang et al. [13] utilize this in the *onion* technique. The first layer of the "onion" comprises the records that belong to the convex hull. Generally, the i-th layer comprises the convex hull records when the first $i-1$ layers are ignored. With k layers pre-computed, *onion* answers top-k queries by looking only across these k layers. We stress that k-skyband and *onion* do not compute minimal subsets, i.e., they retain some records that cannot be in the top-k set for any $\boldsymbol{w}$. Also, they are unable to cater for the constraint expressed by R.

Finally, there is a parallel with spatial queries when the location of the query point is uncertain. For example, [47] and [46] compute all the k nearest neighbors and k reverse nearest neighbors, respectively, when the query point could be anywhere on a line segment. For uncertain points a, b, and r that may lie anywhere in axis-parallel hyper-rectangles A, B, and R, respectively, Emrich et al. [20] determine whether r is definitely closer to a than to b. Techniques in that sphere do not apply to our problem, since they consider spatial proximity, as opposed to score-based ranking.

3. PRELIMINARIES

3.1 Problem Definition

The input of UTK comprises a dataset D, a positive integer k, and a region R. Each record $\boldsymbol{p} \in D$ includes d values, i.e., $\boldsymbol{p} = (x_1, x_2, ..., x_d)$. We assume that higher values are preferable for every attribute. Effectively, the attributes define a d-dimensional *data domain*, where records can be seen as vectors. To cope with datasets of large scale, we assume that D is organized by a spatial index, such as an R-tree [23].

In plain top-k queries, the score of $\boldsymbol{p}$ is derived by an input *weight vector* $\boldsymbol{w} = (w_1, w_2, ..., w_d)$ as $S(\boldsymbol{p}) = \sum_{i=1}^{d} w_i \cdot x_i$. The k records with the highest scores form the top-k result. Without loss of generality, we assume that $w_i \in (0, 1)$ for each $i \in [1, d]$ and that $\sum_{i=1}^{d} w_i = 1$. These conditions do not constrain user preferences, because record ranking does not depend on the magnitude of $\boldsymbol{w}$ but only on its direction [13]. However, they enable dropping one weight. That is, from $\sum_{i=1}^{d} w_i = 1$ we derive $w_d = 1 - \sum_{i=1}^{d-1} w_i$, and thus reduce the domain of $\boldsymbol{w}$ to a $(d-1)$-dimensional space, called the *preference domain*[3]. Henceforth, by $\boldsymbol{w}$ we refer to the $(d-1)$-dimensional form of the weight vector.

The third input to UTK is a region R in preference domain. For ease of presentation, we assume that it is an axis-parallel hyper-rectangle, yet our techniques apply directly to general convex polytopes.

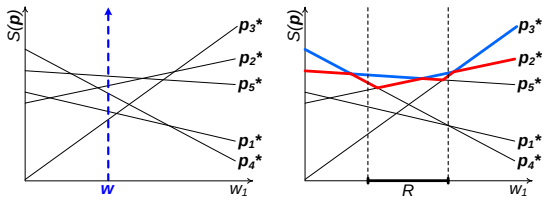
We distinguish two UTK versions. UTK_1 reports the set of exactly those records that may rank among the top-k when the weight vector lies inside R. "Exactly" here means that the reported set is *minimal*, i.e., for every record $\boldsymbol{p}$ in it, there is at least one weight vector in R for which $\boldsymbol{p}$ belongs to the top-k set. The second version, UTK_2, reports the exact top-k set for every possible weight vector in R. While there are infinite possible vectors in R, the output is a partitioning of R, where each partition is associated with the exact top-k set when $\boldsymbol{w}$ lies anywhere inside that partition.

The distinguishing power of score-based ranking diminishes with d [35, 51]. Specifically, as d grows, the scores of all records quickly converge, which renders rank-aware processing (including the standard top-k query) meaningless for more than a handful of dimensions. Therefore, we focus on low-dimensional settings.

3.2 Example and Connection to $\leq k$-level

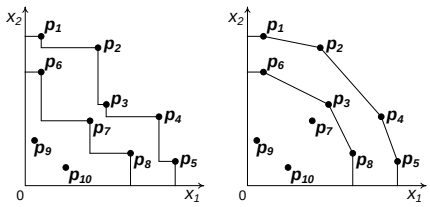
Consider a 2-dimensional D. As discussed above, the preference domain is 1-dimensional, since $w_2 = 1 - w_1$. If we plot the scores of records $\boldsymbol{p}_i \in D$ as functions of w_1, they are

[3]If a vector needs to be mapped from/to the full-dimensional domain, this can be done using the aforementioned equation for w_d. Mapping a convex polytope is performed similarly, by applying the same process to its defining vertices.



(a) Dual space (b) $\leq$ 2-level

Figure 2: Dual space and $\leq k$-level for $d = 2$



(a) 2-skyband (b) 2 *onion* layers

Figure 3: k-skyband and *onion* layers for $d = 2$

each mapped into a line $\boldsymbol{p}_i^*$. Figure 2(a) demonstrates that dual representation for $n = 5$ records. Our (1-dimensional) weight vector $\boldsymbol{w}$ corresponds to a position on the horizontal axis. If we shoot a vertical ray from that position upwards, the ray meets the lines $\boldsymbol{p}_i^*$ in ascending order of $S(\boldsymbol{p}_i)$. I.e., the top-k set for $\boldsymbol{w}$ comprises those records that correspond to the k lines that are met last by the ray. In Figure 2(a), $\boldsymbol{w}$ corresponds to the dashed ray that meets lines $\boldsymbol{p}_2^*$ and $\boldsymbol{p}_5^*$ last, i.e., the top-2 set for $\boldsymbol{w}$ includes $\boldsymbol{p}_2$ and $\boldsymbol{p}_5$.

Effectively, the dual representation maps the dataset into an arrangement of lines. To identify all possible top-k sets (for any $\boldsymbol{w}$ in the preference domain) is equivalent to computing all points that lie on one of the lines and have fewer than k other lines above them. E.g., in Figure 2(b), the blue piecewise linear curve indicates the record that ranks 1-st for any possible $\boldsymbol{w}$ (i.e., for any position of $\boldsymbol{w}$ on the horizontal axis), while the red curve captures the record that ranks 2-nd. The blue curve is called the 1-level of the arrangement, the red is called the 2-level, while together they form the $\leq$ 2-level [5]. In general, the $\leq k$-level captures the top-k set for any $\boldsymbol{w}$. Thus, UTK is dual to a constrained version of $\leq k$-level, where $\boldsymbol{w}$ is bounded in a specific region R of the preference domain. In Figure 2(b), if R is the shown interval, the UTK result for $k = 2$ corresponds to the part of the $\leq$ 2-level that is between the dashed lines. Although we used a 2-dimensional example, the same parallel holds for any d, with records mapped into hyperplanes. Vector $\boldsymbol{w}$ still produces a ray, shot upwards from point $(w_1, w_2, ..., w_{d-1}, 0)$.

The reduction to a constrained version of $\leq k$-level, although conceptually interesting, is not useful in terms of algorithmic design. As elaborated in [4], efficiently computing the $\leq k$-level has not been resolved completely. Specifically, the best known algorithm has a time complexity of $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ and is purely of theoretical interest [36]. In the context of large datasets, the only $\leq k$-level solutions [16, 15] are explicitly for $d = 2$, which is a degenerate case, because the preference domain becomes 1-dimensional.

3.3 Baseline Algorithm

We construct a baseline UTK algorithm using existing methods. As elaborated in Section 2, the k-skyband and the k *onion* layers are supersets of all possible top-k results. Actually, since weights are positive, in each *onion* layer it suffices to keep records that define convex hull facets with norm in the first quadrant. Assuming that $k = 2$, Figure 3(a) illustrates the 2-skyband in a dataset of 10 records. The outer staircase line passes through records that are not dominated by any other, and the inner line via those dominated by exactly one. The 2-skyband is their union, i.e., set $\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_8\}$. Figure 3(b) shows the first $k = 2$ *onion* layers. They include the same records as the 2-skyband except $\boldsymbol{p}_7$. The k layers are always a subset of the k-skyband [38].

Generally, both the k-skyband and *onion* retain some records that cannot be in any top-k set [25]. For instance, record $\boldsymbol{p}_3$ in Figure 3 is in the 2-skyband and in the 2 *onion* layers, but it cannot be in the top-2 set for any $\boldsymbol{w}$. More so in UTK, where $\boldsymbol{w}$ is further bounded by R, the k-skyband and *onion* layers become even looser supersets of the records that could appear in the top-k set.

The baseline uses either the k-skyband or the k *onion* layers as a filtering step, and then determines for each retained candidate $\boldsymbol{p}$ whether it is part of the UTK result. The latter task can be accomplished by a monochromatic reverse top-k query at $\boldsymbol{p}$; we use the $kSPR$ methodology [45], constrained to consider only region R. If $kSPR$ reports an empty set, $\boldsymbol{p}$ is disqualified. Otherwise, it is part of the UTK result. In UTK_1, we may terminate $kSPR$ as soon as the first sub-region of R is found where $\boldsymbol{p}$ is in the top-k set. In UTK_2, however, $kSPR$ is left to terminate regularly, so as to produce all sub-regions of R where $\boldsymbol{p}$ belongs to the top-k set (note that for this problem variant the baseline's output will have a different, yet semantically equivalent form to the UTK_2 output described in Section 3.1).

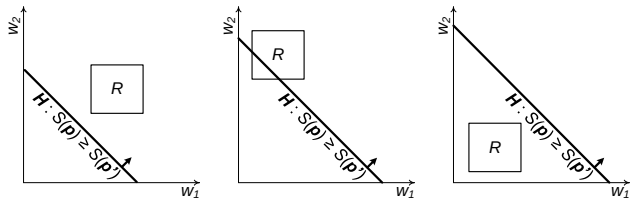
We evaluate the k-skyband and the *onion* variants of the baseline in the experiments. Implementation-wise, to compute *onion* layers, it is beneficial [10, 52] to first compute the k-skyband (e.g., by BBS) and then iteratively run *quickhull* [7] to derive *onion* layers off the k-skyband one by one.

4. R-SKYBAND ALGORITHM

In this section, we present the *r-skyband algorithm* (RSA) for the UTK_1 problem. A key notion is *r-dominance*. The intuition is that, in traditional dominance, the dominator is preferable to the dominee for any weight vector $\boldsymbol{w}$; r-dominance, instead, is specific to weight vectors in R. Although a record may not dominate another in the traditional sense, it might always score higher when $\boldsymbol{w}$ is bounded in R.

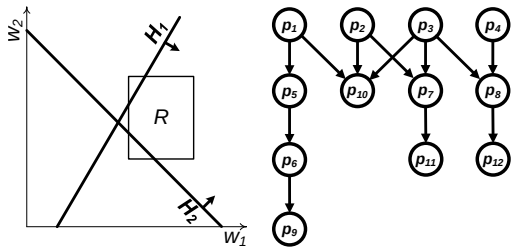
Definition 1. Given a region R in the preference domain, we say that record $\boldsymbol{p}$ r-dominates another record $\boldsymbol{p}'$ when $S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$ for any weight vector in R, and there is at least a weight vector in R for which $S(\boldsymbol{p}) > S(\boldsymbol{p}')$.

Consider two records, $\boldsymbol{p}$ and $\boldsymbol{p}'$, which are *incomparable* in terms of traditional dominance, i.e., none dominates the other. The equality $S(\boldsymbol{p}) = S(\boldsymbol{p}')$ corresponds to a hyperplane in the preference domain, while the inequality $S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$ corresponds to a half-space. We distinguish 3 cases regarding the positioning of that half-space w.r.t. R, shown in Figure 4. In the first case (Figure 4(a)), $\boldsymbol{p}$ r-dominates $\boldsymbol{p}'$ because R is fully covered by half-space $\boldsymbol{H} : S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$, i.e., $\boldsymbol{p}$ scores higher for any $\boldsymbol{w} \in R$. In the second case (Figure 4(b)), there is a part of R where $\boldsymbol{p}$ scores higher than $\boldsymbol{p}'$ and another where it is the other way around.



(a) $\boldsymbol{p}$ r-dominates (b) r-incomparable (c) $\boldsymbol{p}$ r-dominated

Figure 4: r-dominance cases for records $\boldsymbol{p}$ and $\boldsymbol{p}'$



(a) r-skyband issue (b) r-dominance graph

Figure 5: r-dominance, r-skyband and r-dominance graph

In this case, none of the records r-dominates the other, and we call them *r-incomparable*. The third case (Figure 4(c)) is symmetric to the first, with $\boldsymbol{p}$ being r-dominated by $\boldsymbol{p}'$, i.e., scoring lower than $\boldsymbol{p}'$ anywhere in R.

Observe that in the above investigation we considered two incomparable records, i.e., records for which traditional dominance could not make safe conclusions about which ranks higher. In the first and in the third case, however, the notion of r-dominance allowed such conclusions to be safely made. Unlike traditional dominance, it is not straightforward to determine whether a record $\boldsymbol{p}$ r-dominates another record $\boldsymbol{p}'$. We test them by checking whether all vertices that define R are inside or outside $\boldsymbol{H}: S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$. If they are all inside (outside), $\boldsymbol{p}$ r-dominates (is r-dominated by) $\boldsymbol{p}'$. Otherwise, the records are r-incomparable. The containment test for each vertex takes $O(d)$ time, thus a total of $O(md)$ for the r-dominance test, where m is the number of vertices that define R.

4.1 Filtering Step (r-Skyband)

The r-dominance concept is fundamental to several aspects of *RSA*. The first is that, from Definition 1, we infer that any record that is r-dominated by k or more other records cannot be in the UTK_1 result, and can thus be eliminated from consideration. Formally, the only records that could be in the UTK_1 result must belong to the *r-skyband*:

Definition 2. The r-skyband of a dataset includes only those records that are r-dominated by fewer than k others.

The r-skyband is a subset of the traditional k-skyband. To see this, recall that a record may r-dominate another even if they are incomparable in the traditional sense.

Another important fact is that the r-skyband is still a superset of the UTK_1 result. The reason is that it relies on pairwise r-dominance, and is unable to capture the combined effect of multiple competitors on the ability of a specific record to enter the top-k set. To exemplify, consider record $\boldsymbol{p}$ and two of its competitors, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, in a UTK_1 scenario where $k = 1$. Assume that half-spaces $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ in Figure 5(a) correspond to $S(\boldsymbol{p}_1) \geq S(\boldsymbol{p})$ and $S(\boldsymbol{p}_2) \geq S(\boldsymbol{p})$, respectively. None of the half-spaces covers the entire R, thus $\boldsymbol{p}$ is not r-dominated by $\boldsymbol{p}_1$ or $\boldsymbol{p}_2$. However, there is no part of R that is not covered by any of the two half-spaces. In other words, although $\boldsymbol{p}$ is a member of the r-skyband, it does not belong to the UTK_1 result. Thus, the r-skyband can only be used as a filtering tool for UTK_1 processing.

The computation of r-skyband (using the index on D) is similar to k-skyband computation, and follows the *BBS* paradigm, with a few important differences. First, instead of traditional dominance, we use r-dominance; the r-dominance test described previously applies both for record-to-record and for record-to-MBB dominance testing (by representing the MBB by its top corner, as per normal). Second, we use a different visiting order for R-tree nodes and records, i.e., a different sorting key for the search heap. We exploit the fact that $\boldsymbol{w}$ is bounded in R to determine a more effective order, i.e., to expedite the search by guiding it first to the most likely members of the r-skyband.

Specifically, our adapted *BBS* uses a max-heap, where the sorting key is the score (of records or of the top corner of node MBBs) according to the *pivot* vector of R. The pivot's value in each dimension is the average value of R's vertices in that dimension. Due to the convexity of R, the pivot is guaranteed to lie in R [8]. Other than that, the *BBS* process is the same as described in Section 2, by simply replacing traditional dominance with r-dominance tests.

Recall that *BBS* can use any monotone metric as key for its search heap (instead of distance to the top corner of the data domain), as long as the metric guarantees that any record $\boldsymbol{p}'$ popped from the heap (i.e., considered for inclusion into the k-skyband) after a record $\boldsymbol{p}$ cannot dominate $\boldsymbol{p}$. That requirement is upheld in our adaptation. In particular, since records are popped from the search heap in decreasing score order according to the pivot, any record $\boldsymbol{p}'$ popped after $\boldsymbol{p}$ has smaller score according to at least one weight vector in R (i.e., the pivot), and thus cannot r-dominate $\boldsymbol{p}$.

In addition to filtering (by means of the r-skyband), r-dominance is also useful in the refinement step, as we will see in Section 4.2. Thus, we record all pair-wise r-dominance relationships between members of the r-skyband. The data-structure to store these relationships is a directed acyclic graph (DAG), called the *r-dominance graph* G. Using a DAG to maintain dominance relationships is common in the skyline literature (e.g., [54, 32]). Figure 5(b) illustrates G in an example where $k = 4$. An arc from node (record) $\boldsymbol{p}$ to $\boldsymbol{p}'$ encodes the fact that $\boldsymbol{p}$ r-dominates $\boldsymbol{p}'$. Note that an arc from $\boldsymbol{p}_1$ to $\boldsymbol{p}_6$ or to $\boldsymbol{p}_9$ is not needed, because this is already implied by the transitivity of the r-dominance relationship.

Building G does not require any r-dominance tests, other than those already performed for r-skyband computation. In particular, prior to inserting any new record $\boldsymbol{p}$ into the r-skyband, *BBS* anyway needs to identify all existing r-skyband members that r-dominate it. Also, due to pivot-based sorting, these are the only records that could possibly r-dominate $\boldsymbol{p}$. Thus, at the time of inclusion to the r-skyband, we know already all the records that r-dominate $\boldsymbol{p}$. Their number is called the *r-dominance count* of $\boldsymbol{p}$.

4.2 Refinement Step

Compared to the baseline approach in Section 3.3, we have established that *RSA* considers a subset of its candidates.

However, the most major performance improvements are achieved by its refinement process. *RSA* considers candidates one by one, in decreasing order of their r-dominance counts. The reason behind this order is that if a candidate (node of G) is confirmed to be part of the UTK_1 result, by Definition 1, so do all its ancestors in G, thus reducing the number of candidates to be verified.

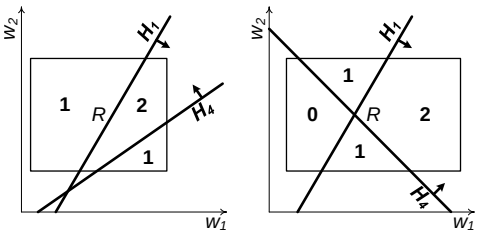
The verification of individual candidates may further benefit from the r-dominance relationships stored in G. Consider Figure 5(b), where $k = 4$. The first candidates to verify are $\boldsymbol{p}_9, \boldsymbol{p}_{10}, \boldsymbol{p}_{11}, \boldsymbol{p}_{12}$, with r-dominance count 3. Ties are resolved arbitrarily. Assume that $\boldsymbol{p}_{11}$ is chosen to be verified first. Any of its ancestors in G is known to r-dominate it. Thus, $\boldsymbol{p}_{11}$'s verification may ignore $\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_7$, and simply reduce its *rank quota* to $k - 3 = 1$; that is, with 3 ancestors ignored, it is as if $\boldsymbol{p}_{11}$ is to be verified for a UTK_1 query with parameter k reduced by 3. All remaining competitors are r-incomparable to $\boldsymbol{p}_{11}$.

With ancestors ignored, the verification of a candidate $\boldsymbol{p}$ entails partitioning R by half-spaces $\boldsymbol{H}_i : S(\boldsymbol{p}_i) \geq S(\boldsymbol{p})$, each corresponding to one of the (remaining) competitors $\boldsymbol{p}_i$. Formally, the supporting hyperplanes of these half-spaces define an arrangement bounded by R. Every cell (i.e., partition) in that arrangement lies inside a set of half-spaces. The competitors that correspond to these half-spaces are exactly those that rank higher than $\boldsymbol{p}$ if $\boldsymbol{w}$ lies in that partition. If any of the partitions in R is inside fewer half-spaces than the candidate's rank quota, then the candidate (and, consequently, each of its ancestors in G) is verified. Otherwise, it is not part of the UTK_1 result.

The problem with this approach is that computing the arrangement of all half-spaces $\boldsymbol{H}_i$ in R is a costly process (requiring $O(n^d)$ time [5], where n is the number of competitors). Instead, we initialize an empty arrangement in R, and insert into it half-spaces for just a small, carefully selected subset of competitors, hoping to (safely) verify or disqualify the candidate $\boldsymbol{p}$ without having to consider the rest. The competitors chosen are those with the smallest r-dominance count. The rationale is that these competitors are intuitively the strongest, and in turn, the most likely to help disqualify the candidate. Note that the r-dominance counts here should ignore the candidate's ancestors (because, as stated before, they are disregarded during the verification of $\boldsymbol{p}$).

In our running example on the verification of $\boldsymbol{p}_{11}$, the competitors with the smallest r-dominance count (of 0) are $\boldsymbol{p}_1$ and $\boldsymbol{p}_4$. Their respective half-spaces, $\boldsymbol{H}_1$ and $\boldsymbol{H}_4$, are inserted into the arrangement. Assume that Figure 6(a) shows the resulting arrangement. The count in each partition indicates how many of the inserted half-spaces cover that partition. Since all partitions in R meet or exceed $\boldsymbol{p}_{11}$'s rank quota (i.e., 1), the candidate is disqualified without a need to consider any other competitor. Observe that we consider only the arrangement within R. The rest of the preference domain is irrelevant to UTK_1 processing (and so is the small triangular area bounded downwards by the w_1 axis, whose count would have been 0 if it were inside R). Note also that when a candidate is disqualified, it is directly removed from G in order to avoid waste of computations when other candidates are considered; the actual UTK_1 records are enough to disqualify any non-UTK_1 candidate.

Figure 6(b) illustrates an alternative scenario where the arrangement produced by $\boldsymbol{H}_1$ and $\boldsymbol{H}_4$ includes a *promising partition*, i.e., a partition with count smaller than the rank quota of the candidate. The promising partition in our case has count 0 (recall that $\boldsymbol{p}_{11}$'s rank quota are 1). This partition could render $\boldsymbol{p}_{11}$ a member of the UTK_1 result. The question is whether any of the remaining competitors could produce a half-space that covers or overlaps with the promising partition. Lemma 1 provides the answer.



(a) Counts 1, 2, 1 (b) Counts 0, 1, 1, 2

Figure 6: Arrangement and partitions in R

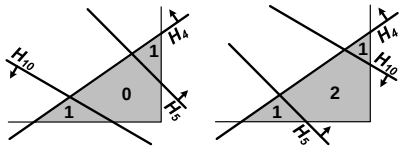
LEMMA 1. *Let* $\mathbf{p}_i$ *be a competitor of the candidate* $\mathbf{p}$*, whose half-space* $\mathbf{H}_i$ *has already been inserted into the arrangement. Let also* $\mathbf{p}_j$ *be a record that is r-dominated by* $\mathbf{p}_i$*, and* ρ *be a partition in the arrangement that is not covered by* $\mathbf{H}_i$*. The candidate* $\mathbf{p}$ *is guaranteed to r-dominate* $\mathbf{p}_j$ *in partition* ρ*.*

PROOF. First, by the definition of half-space $\boldsymbol{H}_i$, anywhere outside $\boldsymbol{H}_i$ it holds that $S(\boldsymbol{p}_i) < S(\boldsymbol{p})$. Second, from the definition of r-dominance, anywhere inside R it holds that $S(\boldsymbol{p}_j) \leq S(\boldsymbol{p}_i)$. Combining the two facts, we deduce that $S(\boldsymbol{p}_j) < S(\boldsymbol{p})$ for every partition ρ of the arrangement that is outside $\boldsymbol{H}_i$, i.e., $\boldsymbol{p}$ r-dominates $\boldsymbol{p}_j$ in ρ. □

Returning to our example, the answer to whether the half-space of any of the remaining competitors could cover or overlap with the promising partition in Figure 6(b) is negative. The reason is that the promising partition lies outside both $\boldsymbol{H}_1$ and $\boldsymbol{H}_4$, while all remaining competitors are r-dominated by $\boldsymbol{p}_1$ or $\boldsymbol{p}_4$. By Lemma 1, we are certain that the count in that partition could not increase further and, hence, we are able to confirm $\boldsymbol{p}_{11}$ (and all its ancestors in G) as part of the UTK_1 result.

In both previous examples, no additional competitors (i.e., other than $\boldsymbol{p}_1$ and $\boldsymbol{p}_4$) needed to be examined to decide on the candidate. To generalize, take Figure 6(a) as an example, but assume that the rank quota of the candidate $\boldsymbol{p}_{11}$ are 2. In this case, there are two promising partitions, both with count 1. Consider, for instance, the bottom-right promising partition. Although it lies outside $\boldsymbol{H}_4$ (and thus all competitors r-dominated by $\boldsymbol{p}_4$ cannot increase the partition's count), it is inside $\boldsymbol{H}_1$. I.e., Lemma 1 cannot disregard the competitors that are r-dominated by $\boldsymbol{p}_1$. To formalize, the examination of a promising partition needs to consider those of the remaining competitors that are not disregarded by Lemma 1. In our example, these are $\boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_9, \boldsymbol{p}_{10}$.

Observe that the examination of the promising partition is effectively the original problem we started with, where: (i) instead of R, the region of interest is now the promising partition at hand; (ii) the rank quota of the candidate are reduced by the count of the partition (e.g., in the revised example in Figure 6(a), where we started off with rank quota of 2 for the candidate $\boldsymbol{p}_{11}$, they are now reduced by the count of the promising partition (i.e., 1) to 2-1 = 1); (iii) the competitors to consider are those that are not ancestors of the



(a) Counts 1, 0, 1 (b) Counts 1, 2, 1

Figure 7: Local arrangement in promising partition

candidate and are not disregarded by Lemma 1. Therefore, we recursively apply the same process as we did for R.

That is, among the remaining competitors (i.e., $\boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_9, \boldsymbol{p}_{10}$), we choose those with the smallest r-dominance count, and insert their half-spaces into a newly initialized, local arrangement, specific to the promising partition. Note that the r-dominance count now ignores any competitor that is either disregarded or has already been considered. For instance, this means that the r-dominance of $\boldsymbol{p}_{10}$ by $\boldsymbol{p}_2, \boldsymbol{p}_3$ (disregarded) and by $\boldsymbol{p}_1$ (already considered in the original arrangement in R) should be ignored, rendering an r-dominance count of 0 for $\boldsymbol{p}_{10}$. The r-dominance counts of $\boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_9$ are 0, 1, 2, respectively. Among the 4 competitors, $\boldsymbol{p}_{10}$ and $\boldsymbol{p}_5$ (with r-dominance count 0) are inserted into a local arrangement for the bottom-right promising partition in Figure 6(a).

Figures 7(a) and 7(b) provide two alternative scenarios after the insertion of $\boldsymbol{H}_{10}$ and $\boldsymbol{H}_5$ into the local arrangement. In Figure 7(a), there is a promising partition, which is outside both $\boldsymbol{H}_{10}$ and $\boldsymbol{H}_5$ (recall that the rank quota of $\boldsymbol{p}_{11}$ were reduced to 1 in the shaded partition). From Lemma 1 we infer that no remaining competitor could increase its count. We therefore confirm $\boldsymbol{p}_{11}$ as a member of the UTK_1 result, and terminate the verification process for it.

On the contrary, in the scenario of Figure 7(b), there is no promising partition, so there is no further recursion within the current local arrangement, i.e., search within the shaded triangular region is unfruitful. This however does not necessarily mean that the candidate $\boldsymbol{p}_{11}$ is disqualified, because the recursion in the other promising partition (i.e., the top-left partition in Figure 6(a)) could possibly discover a sub-partition where $\boldsymbol{p}_{11}$ does not exceed its rank quota and qualifies for the UTK_1 result.

An optimization here is that when there are multiple promising partitions, we apply the recursive process to them in decreasing order of their counts, so that the most promising partitions are considered first.

4.3 Drill Optimization

The verification process could be enhanced by a simple, yet effective optimization, called *drill*. The optimization applies to the verification of an individual candidate $\boldsymbol{p}$. When region R is examined (in the initial call of the recursive verification process described previously), or when a promising partition is examined (in any of the following calls of the recursive process), before we consider the half-space arrangement at all, we perform a regular top-k query for a *drill vector*, i.e., a vector inside R or inside the partition, respectively. If the candidate $\boldsymbol{p}$ is in the top-k set, it is directly verified as a member of the UTK_1 result.

The drill vector should be chosen such that (i) it falls within the region/partition at hand, and (ii) the candidate $\boldsymbol{p}$ is likely to rank high for it. An intuitive strategy to achieve this twofold objective is to choose the vector that maximizes the score of $\boldsymbol{p}$ (i.e., a linear expression), subject to the (also linear) constraints imposed by the (convex) region or partition. This can be done using linear programming in $O(c^d d! n)$ time for a constant c, where n is the number of constraints that define the region/partition [8].

Importantly, when we make a drill, i.e., execute a top-k query for the drill vector, we do not refer at all to the dataset D or its index. Instead, our top-k search runs purely on graph G, in order to utilize directly the r-dominance information it holds. The search follows the branch-and-bound paradigm [29]. We say that we *evaluate* a node of G when we compute its score (according to the drill vector). We first evaluate the nodes with r-dominance count 0, and push them into a max-heap that uses score as the key. Then, we iteratively pop nodes from the heap. If the score of the popped node $\boldsymbol{p}$ is among the top-k we have popped so far, we update the temporary top-k set, evaluate $\boldsymbol{p}$'s children in G, and push them into the max-heap[4]. Otherwise, we disregard $\boldsymbol{p}$. The temporary top-k set is finalized when the k-th smallest score in it is no smaller than the score of the last node popped from the heap (or the heap becomes empty).

4.4 Pseudo-code, Correctness, and Analysis

We present the pseudo-code for *RSA* in Algorithm 1, which invokes the recursive *Verify* procedure (Algorithm 2), i.e., the verification process for a given candidate $\boldsymbol{p}$ in a specified region/partition ρ. The last input of *Verify*, i.e., ignore set I, contains records to be ignored by the verification process. These records are also ignored in r-dominance counting in Line 3.

Algorithm 1 *RSA*(dataset D, region R, value k)

```
1: UTK_1 result set T ← ∅
2: Compute the r-skyband of D
3: G ← build r-dominance graph
4: for each unverified p ∈ G in desc. order of r-dom. count do
5:     I ← ancestors of p in G
6:     if Verify(p, R, k − |I|, I) == True then
7:         Insert into T candidate p and all its ancestors in G
8:     else
9:         Remove p from G
10: Return T
```

Algorithm 2 *Verify*($\boldsymbol{p}$, ρ, rank quota r, ignore set I)

```
1: Perform drill for p; if successful, return True
2: A ← empty arrangement in ρ
3: Insert into A a h/space ∀ competitor with r-dom. count= 0
4: for each promising partition ρ_i in A do
5:     if Lemma 1 confirms the count of ρ_i then
6:         Return True
7:     else
8:         S ← competitors with h/spaces inserted in A
9:         I' ← competitors that cannot affect ρ_i by Lemma 1
10:        r' ← count of ρ_i in A
11:        Verify(p, ρ_i, r − r', I ∪ S ∪ I')
12: Return False
```

Lemmas 2 and 3 regard *RSA*'s correctness and complexity.

[4]Since nodes in G may have multiple parents (like $\boldsymbol{p}_{10}$ in Figure 5(b)), some children of $\boldsymbol{p}$ may already be in the heap via another parent. Only children that are not already in the heap are evaluated/pushed into it.

LEMMA 2. RSA *reports the correct result, i.e., exactly those records that may appear in the top-k set when the weight vector lies inside R.*

PROOF. Records r-dominated by k or more others cannot belong to the UTK_1 result, thus filtering via the r-skyband is safe. Refinement considers all candidates individually, except those that can be safely verified when one of their descendants in G is confirmed to be in the UTK_1 result (which is safe by Definition 1). For each considered candidate, *RSA* constructs an arrangement according to its r-incomparable competitors[5]. If any partition in that arrangement has count smaller than its rank quota, by the definition of half-spaces $\boldsymbol{H}_i$, it means that fewer than k competitors score higher than the candidate in that partition, hence, it is correctly reported in the UTK_1 result. Unless such a partition is discovered, the candidate is disqualified, i.e., *RSA* reports a tight UTK_1 result (not a superset of it). Our arrangement computation (and thus the verification process) for a given candidate $\boldsymbol{p}$ is correct, because the only competitors it ignores are (i) $\boldsymbol{p}$'s ancestors in G, which are already taken into account by appropriate reduction of its rank quota, (ii) those safely disregarded by Lemma 1, and (iii) those whose verification has previously failed. □

LEMMA 3. *The time complexity of* RSA *is* $O(n'^{d+1})$*, where* n' *is the number of candidates received from the filtering step.*

PROOF. The major determinant of *RSA*'s time requirements is the construction of arrangements in its refinement step. In the worst case, the verification process for a candidate $\boldsymbol{p}$ will need to consider all competitors, i.e., it will need to construct the complete arrangement of n' half-spaces, in $O(n'^d)$ time [5]. Since there are n' candidates, the time complexity of *RSA* is $O(n'^{d+1})$. □

The number of candidates received from the filtering step (i.e., n') is no greater than the number of records in the k-skyband of D. There are several analyses on the cardinality of the k-skyband [22, 14, 53], which is generally much smaller than dataset cardinality n (i.e., $n' \ll n$).

4.5 Implementation: Arrangement Indexing

Our discussion so far used half-space arrangements and operations on them as building blocks, abstracting their maintenance and management. In recent database literature there are two approaches to index/maintain a half-space arrangement. One is to index the preference domain using a space-partitioning method, like a Quad-tree [50, 35]. This approach computes the exact geometry of each arrangement cell (i.e., partition, in our UTK_1 terminology) and records it inside all the quads that it overlaps with. Another approach is to use an implicit representation of the partitions by indicating only the half-spaces that define them (rather than computing and storing their explicit geometric representation) and to organize the partitions using a binary tree [45]. The leaves of the binary tree are divided when a new half-space is inserted, increasing the tree height by 1. Both approaches support the incremental insertion of half-spaces into the arrangement, as well as the counting and the identification of half-spaces that cover a partition. Our methodology can be implemented using either of the two approaches, but we adopt the latter as it performs better.

An important remark is that in all aforementioned studies (using either the Quad-tree or the binary tree approach) a single arrangement and a single index were maintained while a query executed. In our case, there are multiple indices built and disposed during execution. Specifically, an index is built for every local arrangement considered, i.e., for every call of the recursive process *Verify* (Algorithm 2). This much smaller index is discarded when the recursion proceeds to the promising sub-partitions of the local arrangement (if any). The smaller index size implies smaller computation cost for operations on the local arrangement (e.g., half-space insertion, coverage counting, etc), while its prompt discarding reduces memory consumption. Note that this is not only an implementation detail, but an optimization made possible by our algorithmic design, which processes each partition irrelevantly from others, thus allowing a focusing only on those select half-spaces that may affect the local arrangement. That flexibility is not possible in the single arrangement, single index approach of previous studies. Furthermore, many complex index optimizations used in both the Quad-tree and the binary-tree approaches (e.g., pertaining to reduction of space requirements) are no longer necessary with our multiple, small, and disposable indices.

5. JOINT ARRANGEMENT ALGORITHM

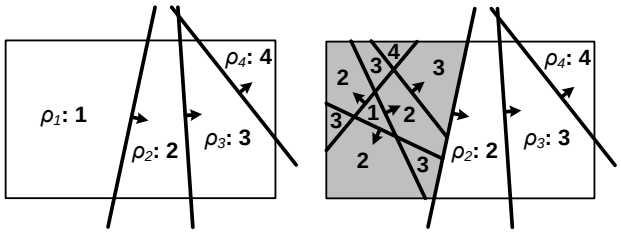
In this section, we present the *Joint Arrangement Algorithm* (*JAA*) for the UTK_2 problem. Its filtering step is the same as *RSA*'s, but refinement is fundamentally different. The output of *JAA* is a single partitioning of R, where each partition is associated with the respective top-k set. Note that UTK_2 has a more explicit/detailed output than UTK_1.

The key idea is that *JAA* works on a *common global arrangement* (i.e., partitioning of R) for *all* candidates received from the filtering step, as opposed to verifying or disqualifying individual candidates via independent, candidate-specific arrangements. When *JAA* terminates, the common global arrangement is returned as output. A central element in *JAA* is the choice of an *anchor* record among the candidates. At first, the partitioning of R is performed according to that anchor, yet the anchor for some partitions (in the common global arrangement) may change to another candidate in some cases, which may also recursively re-occur for their own sub-partitions.

Refinement in *JAA* commences by choosing as anchor record $\boldsymbol{p}$ one of the candidates received from the filtering step. The anchor choosing strategy is essential and will be elaborated in Section 5.1. Using the anchor in a role similar to that of a candidate for verification, we apply the same verification process as in Section 4.2. Importantly, however, we disable early termination and the drill optimization. That is, we do not halt the process when $\boldsymbol{p}$ is found to be part of the top-k set for some partition of R. Instead, we proceed until each partition ρ in R is safely classified as either an *equal-to*, a *less-than*, or a *greater-than* partition.

In the verification-like process for the anchor $\boldsymbol{p}$, the *equal-to* and *less-than* partitions are guaranteed by Lemma 1 to have no overlap with any of the half-spaces induced by the remaining competitors of $\boldsymbol{p}$. That is, the rank k' of $\boldsymbol{p}$ in any *equal-to* or *less-than* partition ρ is confirmed, and it is equal to the number of already inserted half-spaces that cover ρ, plus 1. In *equal-to* partitions $k' = k$, that is, we know exactly

[5]Local arrangements at the leaves of the recursion tree of *Verify*, together form an implicit arrangement in entire R.



(a) Partitioning R; anchor $\boldsymbol{p}$ (b) Partitioning ρ_1; anchor $\boldsymbol{p}'$

Figure 8: Common global arrangement in JAA ($k = 2$)

the top-k set; recall from Section 4.5 that the arrangement index maintains not only the number, but also the identity of the half-spaces $\boldsymbol{H}_i$ that cover each partition, and thus the specific candidates $\boldsymbol{p}_i$ that contribute to k'. No further processing is required for *equal-to* partitions, and they are already finalized in JAA's common global arrangement. On the other hand, in *less-than* partitions, k' is smaller than k. That is, we know the k'-member prefix of the top-k set (which includes the anchor $\boldsymbol{p}$) but not its last $k-k'$ members.

The third type of partitions are *greater-than*. The number of half-spaces that cover such partitions is $k' > k$. That is, we are certain that the anchor $\boldsymbol{p}$ does not belong to the top-k set for them. Note that in the case of *greater-than* partitions, k' does not need to be confirmed by Lemma 1, meaning that as long as k half-spaces are found to cover a partition, we are certain that $k' > k$, and may safely classify the partition as *greater-than*.

To exemplify, assume that refinement commences using as the initial anchor a record $\boldsymbol{p}$, which has 3 competitors with dominance count 0. The half-spaces that correspond to these competitors produce the arrangement (partitioning of R) shown in Figure 8(a). The number in each partition corresponds to k', i.e., the number of half-spaces that cover the partition plus 1. Assume that $k = 2$, and that Lemma 1 guarantees that k' is confirmed for ρ_1 and ρ_2, i.e., none of the remaining competitors of $\boldsymbol{p}$ could induce half-spaces that have any overlap with these partitions. ρ_2 is an *equal-to* partition, where we know the entire top-k set. ρ_1 is a *less-than* partition, where we know that $\boldsymbol{p}$ is the top-1 record, but do not know the other member of the top-k set. Value k' for ρ_3 and ρ_4 is already greater than k, thus they are *greater-than* partitions (regardless of whether k' is confirmed by Lemma 1), and we know that the anchor $\boldsymbol{p}$ is not part of the top-k set. The verification-like process for the current anchor $\boldsymbol{p}$ culminates at that point, since all partitions are successfully classified as *equal-to*, *less-than*, or *greater-than*.

Further processing is required in *less-than* and *greater-than* partitions. Consider the former case first. For each *less-than* partition ρ, we know only the k' top-scoring members of the top-k set (where $k' < k$) and we need to compute the next $k - k'$. That is, we need to recursively solve an instance of the UTK_2 problem, using $k - k'$ instead of k, using ρ instead of R, and ignoring the k' candidates known already to be the top-scorers anywhere inside ρ. Since the initial anchor $\boldsymbol{p}$ is among the k' ignored candidates, the verification-like process in ρ should use a different anchor record $\boldsymbol{p}'$ (using the strategy described in Section 5.1). In turn, the verification-like process for $\boldsymbol{p}'$ in ρ culminates when all the sub-partitions of ρ are successfully classified as *equal-to*, *less-than*, or *greater-than*, with further recursive processing required for the latter two types.

Continuing the example in Figure 8(a), to deal with the *less-than* partition ρ_1, we select a new anchor $\boldsymbol{p}'$ for it, and execute a verification-like process for $\boldsymbol{p}'$ in ρ_1. The known prefix of the top-k set includes the former anchor $\boldsymbol{p}$, thus the rank quota of $\boldsymbol{p}'$ are $k - 1 = 1$. Figure 8(b) demonstrates the produced partitioning of ρ_1 (shown shaded), and its place in the common global arrangement of JAA. Since the rank quota of $\boldsymbol{p}'$ are 1, among the sub-partitions of ρ_1, one is *equal-to* (thus finalized in JAA's common global arrangement) and all the rest are *greater-than*.

Having elaborated on *less-than* partitions, we next focus on the *greater-than* type. For each *greater-than* partition ρ, we know only that the initial anchor $\boldsymbol{p}$ cannot be part of the top-k set. Therefore, we choose a new anchor $\boldsymbol{p}'$ for ρ, and apply recursively the verification-like process for $\boldsymbol{p}'$ in ρ. To accelerate this process, however, we ignore the initial anchor $\boldsymbol{p}$, and all its descendants in G, because (they are guaranteed to rank even lower than $\boldsymbol{p}$, and hence) they too cannot belong to the top-k set anywhere in ρ. In Figure 8(b), for instance, we choose a new anchor for each of the *greater-than* partitions ρ_3 and ρ_4, and partition them independently, ignoring $\boldsymbol{p}$ and all its descendants in G.

JAA terminates when all partitions in the common global arrangement are *equal-to* (for some anchor). The common global arrangement is then reported as the UTK_2 solution.

5.1 Anchor Choosing Strategy

The performance of JAA depends on the anchor choosing strategy. Consider, for example, the handling of *less-than* partition ρ_1 in Figure 8(a). If the new anchor chosen for ρ_1 is not among the top-k for any weight vector in ρ_1, the verification-like process is bound to produce only *greater-than* sub-partitions. That is, the partitioning of ρ_1 will fail to finalize any sub-partition in the common global arrangement, and instead will leave us with multiple *greater-than* sub-partitions, each requiring further processing. In other words, a poorly chosen anchor not only fails to facilitate the JAA process, but is actually detrimental to it.

To avoid this, when we choose the anchor for a partition ρ, we need to ensure that it belongs to the top-k set for at least one weight vector in ρ. Furthermore, among the members of the top-k set for that vector, intuitively, we want to choose the k-th (i.e., the lowest-scoring). This way we guarantee that at least one of the resulting sub-partitions of ρ will be *equal-to* and will be directly finalized in JAA's common global arrangement. To compute a weight vector that falls in ρ, and in turn select as anchor the k-th scoring record for that vector, we use the drill operation from Section 4.3[6]. Note that we apply the same anchor selection strategy when choosing the initial anchor for region R (in the beginning of the refinement step), using R's pivot as the drill vector.

5.2 Pseudo-code, Correctness, and Analysis

Algorithm 3 summarizes JAA, which in turn relies on the recursive verification-like procedure *Partition* in Algorithm 4. The first input to *Partition* is the anchor record $\boldsymbol{p}$. A note on *Partition* is that, in Lines 4 and 12, the rank r' of $\boldsymbol{p}$ is the count of ρ_i in the local arrangement $\mathcal{A}$ plus 1, i.e., r' is the rank of $\boldsymbol{p}$ in ρ_i if only the half-spaces that were inserted in Line 2 are considered. Moreover, in Line 12 that

[6]The linear programming formulation to compute the drill vector inside ρ, can use an arbitrary optimization function.

rank needs not be confirmed by Lemma 1, meaning that if $r' > r$, we may safely classify ρ_i as a *greater-than* partition.

Algorithm 3 *JAA*(dataset D, region R, value k)

```
1: Compute the r-skyband of D
2: G ← build r-dominance graph
3: Common global arrangement A_global ← empty arr/ment in R
4: Anchor p ← k-th scoring candidate according to pivot of R
5: I_anc ← ancestors of p in G
6: I_desc ← descendants of p in G
7: Partition(p, R, k − |I_anc|, I_anc ∪ I_desc)
8: Return A_global
```

Algorithm 4 *Partition*($\boldsymbol{p}$, ρ, rank quota r, ignore set I)

```
1: A ← empty arrangement in ρ          ▷ Local arrangement in ρ
2: Insert into A a h/space ∀ comp/tor of p with r-dom. count= 0
3: for each partition ρ_i in A do
4:     if rank r' of p is confirmed by Lemma 1 then
5:         if r' = r then                          ▷ equal-to partition
6:             Finalize ρ_i in A_global
7:         else if r' < r then                     ▷ less-than partition
8:             I_top ← the known top-k' set in ρ_i (note: k' < k)
9:             Choose new anchor p' in ρ_i          ▷ Section 5.1
10:            I_desc ← descendants of p' in G
11:            Partition(p', ρ_i, k − k', I_top ∪ I_desc)
12:    else if rank r' of p is > r then  ▷ greater-than partition
13:        I' ← p and all its descendants in G
14:        Choose new anchor p' in ρ_i              ▷ Section 5.1
15:        I_anc ← ancestors of p' in G
16:        I_desc ← descendants of p' in G
17:        Partition(p', ρ_i, k − |I_anc|, I' ∪ I_anc ∪ I_desc)
18:    else                                  ▷ ρ_i cannot be classified
19:        S ← competitors with h/spaces inserted in A
20:        I' ← competitors that cannot affect ρ_i by Lemma 1
21:        r' ← count of ρ_i in A
22:        Partition(p, ρ_i, r − r', I ∪ S ∪ I')
```

Lemmas 4 and 5 regard *JAA*'s correctness and complexity.

LEMMA 4. JAA *reports the correct result, i.e., every partition in the produced common global arrangement is associated with the actual top-k set when the weight vector falls anywhere in that partition.*

PROOF. The correctness proof for the verification-like process (for an anchor in a given region/partition ρ) follows the same lines as Lemma 2, and relies on the fact that no candidate that could score higher than the anchor in ρ is ignored by the process. Since (i) each partition in the final common global arrangement is produced by the verification-like process for some anchor record, (ii) *JAA* terminates when there are only *equal-to* partitions in the common global arrangement (i.e., partitions where we know exactly the k top-scoring candidates), and (iii) the candidates received from the filtering step are a superset of the data records that could appear in any top-k set in R, we deduce that each partition in the reported common global arrangement is associated with the correct top-k set. □

LEMMA 5. *The time complexity of* JAA *is* $O(n'^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$*, where* n' *is the number of candidates received from the filtering step.*

PROOF. The key factor that determines *JAA*'s complexity is the number of (and the time to compute) partitions in the common global arrangement. If all pruning fails, the common global arrangement includes as many partitions as facets in the $\leq k$-level of the dual hyperplanes for the n' candidates (see Section 3.2), i.e., $O(n'^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ [36], which defines *JAA*'s time complexity. □

6. DISCUSSION

So far we have assumed the most common type of top-k query, where the score of record $\boldsymbol{p} = (x_1, x_2, ..., x_d)$ for a weight vector $\boldsymbol{w} = (w_1, w_2, ..., w_d)$ is defined as $S(\boldsymbol{p}) = \sum_{i=1}^{d} w_i \cdot x_i$. Our methods, however, apply directly to a more general class of scoring functions. The only requirement is that $S(\boldsymbol{p})$ is (i) *monotone*[7] to the attributes of $\boldsymbol{p}$, and (ii) *linear* to the weights in $\boldsymbol{w}$. Monotonicity to the data attributes is required by the BBS paradigm for r-skyband computation (as explained in Section 4.1). Linearity to the weights is required so that equality $S(\boldsymbol{p}) = S(\boldsymbol{p}')$ corresponds to a hyperplane in the preference domain, and inequality $S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$ corresponds to a half-space. The latter is central to all half-space oriented techniques in the refinement steps of *RSA* and *JAA*. Examples of applicable scoring functions include $\sum_{i=1}^{d} w_i \cdot x_i^p$ for $p > 0$ (which effectively also covers all weighted L_p norms), and the even more general $\sum_{i=1}^{d} w_i \cdot f_i(\boldsymbol{p})$ for monotone $f_i(\boldsymbol{p})$ functions defined only on data attributes (i.e., independent to $\boldsymbol{w}$).

7. EXPERIMENTS

We evaluate our techniques using synthetic and real datasets. The synthetic are standard benchmarks for preference queries [11], i.e., *Independent* (IND), *Correlated* (COR), and *Anticorrelated* (ANTI). The real ones are *HOTEL* (418,843 records, 4D) [1], *HOUSE* (315,265 records, 6D) [2], and *NBA* (21,960 records, 8D) [3]. All datasets are indexed by R-trees in main memory. The primary performance metric is computation time, but we also report on space overhead. Table 1 presents the experiment parameters, along with tested and default values (in bold). In each experiment, we vary one parameter and keep the rest at their defaults. Every reported measurement is the average of 50 *UTK* queries, for axis-parallel hyper-cubes R randomly generated in the preference domain. The side-length of R is expressed as a percentage σ of the axis length. All methods were implemented in C++. *SK* and *ON* denote the k-skyband and the *onion* variants of the baseline in Section 3.3. For their *kSPR* component, we used the most efficient algorithm (*LP-CTA*) in [45]. All experiments run on a PC with Intel i7-4770 3.4GHZ CPU and 16GB RAM.

Table 1: Experiment parameters

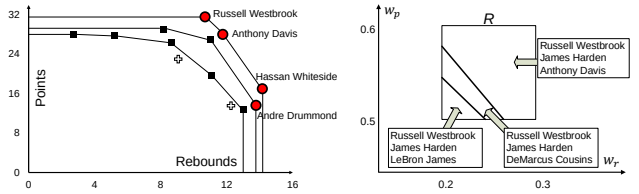
Parameter	Tested values
Dataset cardinality n	100K, 200K, **400K**, 800K, 1600K
Data dimensionality d	2, 3, **4**, 5, 6, 7
Value k	1, 5, **10**, 20, 50, 100
R's side-length σ	0.1%, 0.5%, **1%**, 5%, 10%

7.1 Case Studies and Qualitative Evaluation

We first conduct a couple of *UTK* case studies using real data, and then we present a qualitative comparison, in measurable terms, with traditional operators, like k-skyband, *onion*, and regular top-k queries.

[7]Formally, $S(\boldsymbol{p})$ is monotone if for any pair of records $\boldsymbol{p}, \boldsymbol{p}'$, where $x_i \geq x_i'$ for each $i \in [1, d]$, it holds that $S(\boldsymbol{p}) \geq S(\boldsymbol{p}')$.

In Figure 9, we use actual NBA statistics for the season 2016-2017. On the left (Figure 9(a)), we use $d = 2$ data dimensions, namely, average *Rebounds* and *Points* for each player. The preference domain is 1-dimensional, and corresponds to the weight for *Rebounds*, denoted by w_r. We set $k = 3$ and $R = [0.64, 0.74]$. In the figure, we represent the data domain, and show as red points the players output by UTK_1, i.e., *Russell Westbrook*, *Anthony Davis*, *Hassan Whiteside*, and *Andre Drummond*. The UTK_2 output (not shown) indicates that the top-3 players are the first 3 of them when w_r is in $[0.64, 0.72)$, and the last 3 when w_r is in $[0.72, 0.74]$. For comparison, in Figure 9(a) we also illustrate the first $k = 3$ layers of *onion*, which include the 4 players in the UTK_1 result plus another 7, shown as solid squares. The k-skyband ($k = 3$), which is always a superset of the k *onion* layers, contains the *onion* players plus another 2 shown as crosses, i.e., 13 players in total.



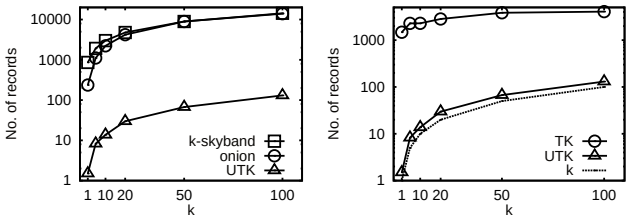
(a) 2D case study (data domain) (b) 3D case study

Figure 9: Case studies on NBA data for season 2016-2017

In Figure 9(b), we use the same NBA data, with an extra dimension, *Assists*. That is, $d = 3$. The preference domain is now 2-dimensional, with axes w_r and w_p (where w_p is the weight for *Points*). We set $k = 3$ and $R = [0.2, 0.3] \times [0.5, 0.6]$, and visualize the UTK_2 output (in the preference domain), i.e., the top-3 set per partition of R. A total of 5 players appear in the UTK result, while *onion* layers and k-skyband contain, respectively, 21 and 25 players (not shown).

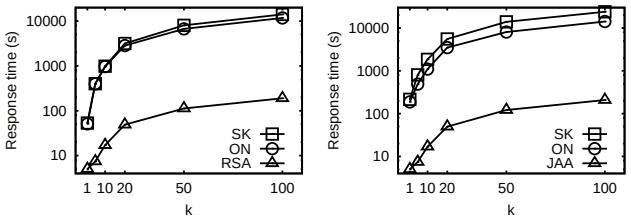
In Figure 10, we look deeper into the relationship between UTK and traditional operators. We use the complete *NBA* dataset, set σ to its default, and vary k, following the standard setup in Table 1. In Figure 10(a), we compare the number of records in the UTK result with that in the first k *onion* layers and in the k-skyband. UTK reports 30 to 100 times fewer records than *onion* and k-skyband. The main reason is that these traditional operators are insensitive to the preferences of the user expressed via region R. Furthermore, even if the R constraint were dropped, *onion* and k-skyband would still not be minimal, i.e., they generally contain some records that cannot belong to the top-k set for any weight vector [25].

Next, using the same data and experiment settings, we examine whether a regular top-k query could simulate UTK_1, i.e., whether a top-k query for a slightly larger k can produce all UTK_1 records. In particular, for each tested value of k, we first processed UTK_1, and then executed an incremental top-k query using for weight vector $\boldsymbol{w}$ the pivot of R (as the most representative vector therein). We probed the top-k query iteratively until it output all records in the UTK_1 result, and we recorded how many records it had to output in total. We plot that number in Figure 10(b) (labeled as TK), together with the number of records in UTK_1 result, and the original value of k for reference. We observe



(a) No. of records (b) Required k

Figure 10: Comparison with traditional operators (NBA)



(a) UTK_1 (b) UTK_2

Figure 11: Effect of k (IND)

that the top-k approach practically required to increase parameter k by 40 to 460 times, and output 30 to 230 times more records than UTK_1 in order to cover its result. These facts confirm that the regular top-k query cannot effectively simulate UTK_1, and that simply increasing k (in a standard top-k query) is not equivalent to UTK_1 processing.

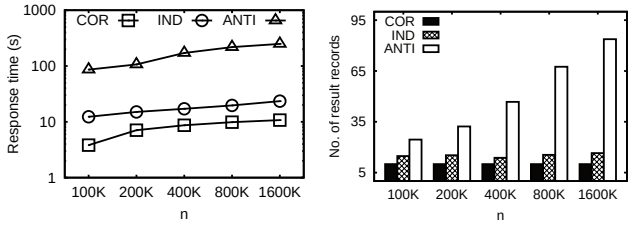
7.2 Performance Evaluation

In the remaining experiments, we focus on performance. In Figure 11, we use IND and study the effect of k on the processing time for UTK_1 and UTK_2. For UTK_1 (in Figure 11(a)), we compare *RSA* with the baselines *SK* and *ON*. For UTK_2 (in Figure 11(b)), we compare *JAA* with the UTK_2 versions of the baselines. As k increases, more records and more possible top-k sets belong to the UTK result. Thus, the processing cost increases too, for all methods. The main observations are:

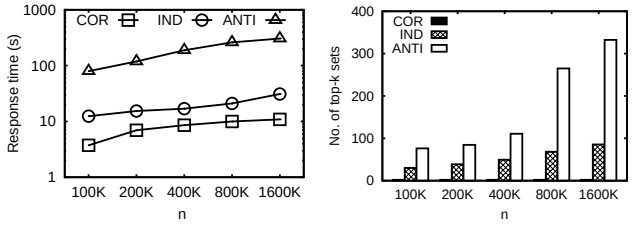
i) Our algorithms (*RSA* and *JAA*) outperform the baselines by one to two orders of magnitude, and the difference grows with k. The reason is twofold; the baselines consider more candidates than our algorithms, and they make numerous calls to their costly $kSPR$ building block.

ii) *ON* performs better than *SK* for both problem versions, because of its tighter, convex-hull-based filtering. In UTK_2 the baselines take almost double the time than in UTK_1, because (in UTK_2) no early termination is possible for their $kSPR$ calls.

Having established that the baselines are impractical for UTK queries, we henceforth exclude them from the experiments. In Figure 12, we apply *RSA* and *JAA* to three different data distributions, as we vary the dataset cardinality n. We report on response time, and on output size, i.e., the number of records in the UTK_1 output and the number of different top-k sets (equivalently, partitions) in UTK_2. There is a clear correlation between the processing time and the output size in all cases. As expected, the smallest output sizes are observed for COR data (because records that are good in one dimension, tend to be good in all dimensions, i.e., r-dominate almost the entire dataset), and the largest for ANTI (because records that are good in

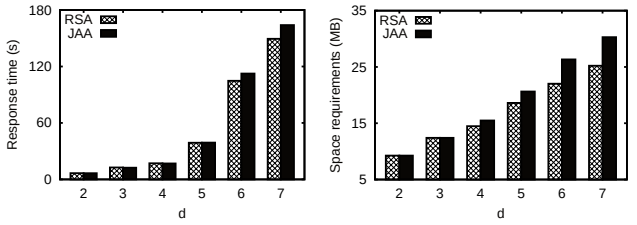


(a) RSA: Response time (b) UTK_1: # result records



(c) JAA: Response time (d) UTK_2: # top-k sets

Figure 12: Effect of n and data distribution



(a) Response time (b) Space requirements

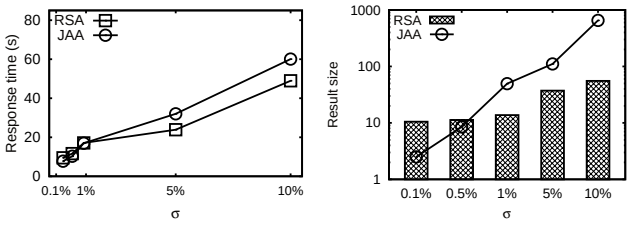
Figure 13: Effect of data dimensionality d (IND)

one dimension, are poor in the others, thus leading to less effective filtering, and more diversity in the possible top-k sets). With regards to n, the response time of our algorithms increases sub-linearly, and remains practical even for the largest datasets (e.g., less than 30s for 1.6M records in IND). The sub-linear increase is in line with the fact that the cardinality of the k-skyband (and therefore of the r-skyband too) is sub-linear to n [22], linking directly to the number of candidates produced by the filtering step.

In Figure 13, we vary the data dimensionality d in IND, and plot the response time and memory requirements of RSA and JAA. Due to its computational geometric nature, the toughness of the UTK problem rises with d. Nonetheless, our algorithms offer practical response times in all cases, taking respectively 149s and 164s in the largest tested dimensionality ($d = 7$). Regarding space requirements, they are in the order of a few MBytes. To substantiate our claims on memory utilization and arrangement indexing in Section 4.5, we mention that for the default $d = 4$, baselines SK and ON need around 10 times more space than our algorithms, primarily due to the single arrangement, single index approach in each of their $kSPR$ calls.

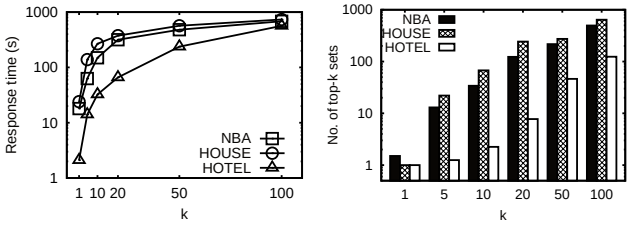
In Figure 14, we vary σ (which determines the size of region R) and present the processing time and result size for RSA and JAA. Result size in RSA refers to number of records in the UTK_1 output, while in JAA to number of partitions (i.e., top-k sets) in the UTK_2 output. As expected, larger R implies larger output, and thus more computations.

Next, we experiment on real data. In the interest of space, we omit the figures for RSA. In Figures 15 and 16, we vary



(a) Response time (b) Result size

Figure 14: Effect of R size (IND)



(a) JAA: Response time (b) UTK_2: # top-k sets

Figure 15: Effect of k (real datasets)

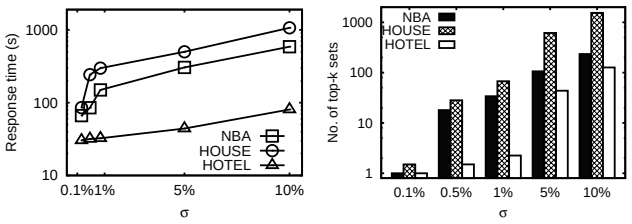
k and σ, respectively, using $HOTEL$, $HOUSE$, and NBA, and measure the response time and output size of JAA. The trends are similar to the synthetic data. Moreover, they establish the correlation between running time and the number of output partitions/possible top-k sets in R. Processing in $HOUSE$ is slower than $HOTEL$, despite similar cardinality, because the former has more attributes (6D versus 4D). Processing in NBA is slower than $HOTEL$ because, although smaller, it has twice as many attributes (8D versus 4D).

8. CONCLUSION

In this paper we study UTK, a novel query that accounts for the inherent inaccuracy in weight specification for top-k processing. With a preference region as input, UTK reports all possible top-k sets when the weight vector may lie anywhere in that region. We formalize the UTK query; draw links to a known problem in computational geometry; distinguish two UTK versions; develop algorithms for their processing; and demonstrate the practicality and scalability of our algorithms using real and benchmark datasets.

9. ACKNOWLEDGMENTS

Kyriakos Mouratidis was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. Bo Tang was supported by the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284).



(a) JAA: Response time (b) UTK_2: # top-k sets

Figure 16: Effect of R size (real datasets)

10. REFERENCES

[1] *Hotel dataset*, 2017. www.hotels-base.com/.

[2] *House dataset*, 2017. www.ipums.org/.

[3] *NBA dataset*, 2017. www.basketball-reference.com/.

[4] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM J. Comput.*, 27(3):654–667, 1998.

[5] P. K. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of computational geometry*, pages 49–119, 2000.

[6] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[7] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[8] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer, 2008.

[9] A. Blum, J. C. Jackson, T. Sandholm, and M. Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5:649–667, 2004.

[10] C. Böhm and H.-P. Kriegel. Determining the convex hull in large multidimensional databases. In *DaWaK*, pages 294–306, 2001.

[11] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[12] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. C. Wong, and W. Zhan. k-regret minimizing set: Efficient algorithms and hardness. In *ICDT*, pages 11:1–11:19, 2017.

[13] 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.

[14] S. Chaudhuri, N. N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, page 64, 2006.

[15] M. A. Cheema, Z. Shen, X. Lin, and W. Zhang. A unified framework for efficiently processing ranking related queries. In *EDBT*, pages 427–438, 2014.

[16] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Indexing reverse top-k queries in two dimensions. In *DASFAA*, pages 201–208, 2012.

[17] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *PVLDB*, 7(5):389–400, 2014.

[18] P. Ciaccia and D. Martinenghi. Reconciling skyline and ranking queries. *PVLDB*, 10(11):1454–1465, 2017.

[19] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.

[20] 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.

[21] T. Ge, S. B. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *SIGMOD Conference*, pages 375–388, 2009.

[22] P. Godfrey. Skyline cardinality for relational processing. In *FoIKS*, pages 78–97, 2004.

[23] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[24] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD Conference*, pages 673–686, 2008.

[25] 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), 2008.

[26] K. G. Jamieson and R. D. Nowak. Active ranking using pairwise comparisons. In *NIPS*, pages 2240–2248, 2011.

[27] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *KDD*, pages 390–398. ACM, 2008.

[28] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[29] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[30] J. Li and A. Deshpande. Ranking continuous probabilistic datasets. *PVLDB*, 3(1):638–649, 2010.

[31] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[32] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang. Finding pareto optimal groups: Group-based skyline. *PVLDB*, 8(13):2086–2097, 2015.

[33] T. Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[34] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *PVLDB*, 2(1):610–621, 2009.

[35] K. Mouratidis, J. Zhang, and H. Pang. Maximum rank query. *PVLDB*, 8(12):1554–1565, 2015.

[36] K. Mulmuley. On levels in arrangement and voronoi diagrams. *Discrete & Computational Geometry*, 6:307–338, 1991.

[37] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1):1114–1124, 2010.

[38] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[39] P. Peng and R. C. Wong. Geometry approach for k-regret query. In *ICDE*, pages 772–783, 2014.

[40] P. Peng and R. C. Wong. k-hit query: Top-k query with probabilistic utility function. In *SIGMOD Conference*, pages 577–592, 2015.

[41] L. Qian, J. Gao, and H. V. Jagadish. Learning user preferences by adaptive pairwise comparison. *PVLDB*, 8(11):1322–1333, 2015.

[42] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, pages 317–328, 2009.

[43] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[44] 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.

[45] 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.

[46] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse k NN search. *VLDB J.*, 16(3):293–316, 2007.

[47] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[48] 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.

[49] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, pages 1406–1408, 2008.

[50] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. In *SIGMOD Conference*, pages 397–408, 2012.

[51] A. Yu, P. K. Agarwal, and J. Yang. Top-k preferences in high dimensions. *IEEE Trans. Knowl. Data Eng.*, 28(2):311–325, 2016.

[52] J. Zhang, K. Mouratidis, and H. Pang. Global immutable region computation. In *SIGMOD Conference*, pages 1151–1162, 2014.

[53] Z. Zhang, Y. Yang, R. Cai, D. Papadias, and A. K. H. Tung. Kernel-based skyline cardinality estimation. In *SIGMOD Conference*, pages 509–522, 2009.

[54] L. Zou and L. Chen. Pareto-based dominant graph: An efficient indexing structure to answer top-k queries. *IEEE Trans. Knowl. Data Eng.*, 23(5):727–741, 2011.