6-2014

Global Immutable Region Computation

Jilian ZHANG
Singapore Management University, jilian.z.2007@smu.edu.sg

Kyriakos MOURATIDIS
Singapore Management University, kyriakos@smu.edu.sg

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

Citation

ZHANG, Jilian; MOURATIDIS, Kyriakos; and PANG, Hwee Hwa. Global Immutable Region Computation. (2014). *SIGMOD '14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data: June 22-27, 2014, Snowbird, UT*. 1151-1162.
Available at: https://ink.library.smu.edu.sg/sis_research/2223

Global Immutable Region Computation

Jilian Zhang Kyriakos Mouratidis HweeHwa Pang
School of Information Systems
Singapore Management University
80 Stamford Road, Singapore 178902
{jilian.z.2007, kyriakos, hhpang}@smu.edu.sg

ABSTRACT

A top-k query shortlists the k records in a dataset that best match the user's preferences. To indicate her preferences, the user typically determines a numeric weight for each data dimension (i.e., attribute). We refer to these weights collectively as the *query vector*. Based on this vector, each data record is implicitly mapped to a score value (via a weighted sum function). The records with the k largest scores are reported as the result. In this paper we propose an auxiliary feature to standard top-k query processing. Specifically, we compute the maximal locus within which the query vector incurs no change in the current top-k result. In other words, we compute all possible query weight settings that produce exactly the same top-k result as the user's original query. We call this locus the *global immutable region* (GIR). The GIR can be used as a guide to query vector readjustments, as a sensitivity measure for the top-k result, as well as to enable effective result caching. We develop efficient algorithms for GIR computation, and verify their robustness using a variety of real and synthetic datasets.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management - Database applications

Keywords

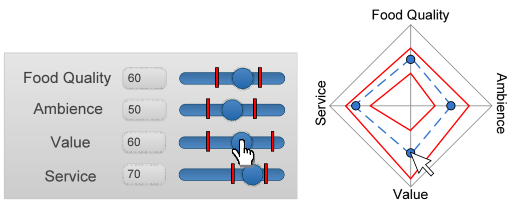
Top-k Search; Sensitivity Analysis

1. INTRODUCTION

Consider a service like HungryGoWhere.com or Yelp.com, where users rate and search for restaurants. The former, for instance, maintains for each registered restaurant the average user ratings in terms of *food quality*, *ambience*, *value for money*, and *service*. Users looking for restaurants base their decisions on these four factors, yet different users weigh each factor differently.

A user interested in dining options can provide a numeric weight for each decision factor and request for a personalized recommendation of, say, the top-10 restaurants according to her preferences. Through those weights, which we collectively refer to as the *query vector*, each restaurant is implicitly associated with a score value, computed as the weighted sum of its four average ratings. The online service may employ an off-the-shelf top-k processing algorithm to report the 10 restaurants with the largest scores.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2610508.


Food Quality 60
Ambience 50
Value 60
Service 70

(a) Text input and slide-bars (b) Radar chart

Figure 1: Weight input and GIR-induced bounds

In this work we propose to supplement the top-k result with a *global immutable region* (GIR). The GIR indicates all the possible weight settings for which the current top-k recommendation holds. For the common case of linear scoring functions, the GIR is a convex polytope in query space, wherein the query vector may freely shift without inducing any changes in the result. In our restaurant example, the query space involves four dimensions, each corresponding to the weight for a factor, e.g., the first axis refers to the weight w_1 for food quality, the second to the weight w_2 for ambience, etc; the GIR is a 4-dimensional polytope in that space. The GIR can be used to guide weight readjustment, for the purpose of sensitivity analysis, as well as for effective top-k result caching.

Suppose that the user in our restaurant example requests for a top-10 recommendation, using an interface like the text-boxes/slide-bars in Figure 1(a) or the radar chart in Figure 1(b), which captures preferences in the form of movable locations on each of the four axes. Assuming weights in the range from 0 to 100, the user requests for a top-10 recommendation by specifying query vector $q = (60, 50, 60, 70)$ – this implies a weight $w_1 = 60$ for food quality, $w_2 = 50$ for ambience, etc. Should the user decide to explore alternative recommendations, she may change the weights and reissue the query. Primarily, she would want to avoid a blind readjustment that induces no change in the top-10 result. At the same time, she would need a sense of how drastically each weight affects the recommendation, so as to avoid overly radical changes. Using the GIR, we may derive a lower and an upper bound mark on each slide-bar (like those shown in Figure 1(a)) in between which the corresponding weight value induces no change in the result. Furthermore, we can inform the user what the new result will be at each of these bounds. Figure 1(b) represents the same bounds in the form of an inner and an outer solid polygon that connect the "tipping points" on the four axes.

GIR computation finds application in sensitivity analysis as well. Effective decision support involves providing the user with both a recommendation, and a measure of its robustness [13, 28]. For example, a robust top-k result would offer the user higher confidence in her decision, while a sensitive one would trigger deeper deliberation. An intuitive robustness measure for a top-k result is the ratio of the GIR volume to the volume of the entire query space. This ratio determines the probability that a randomly and uniformly generated query vector would have the same top-k result as the user's query. The higher this probability, the more robust the result. This measure of robustness was proposed in [30], without however considering efficient approaches to compute it.

Another application of GIR computation is result caching. Suppose that previous top-k results are maintained, along with their GIRs. If the query vector of a new request falls within the GIR of a cached result, the latter can be directly reported. Even if k in the new query is larger than that of the cached result, it is still desirable to report the available (highest-scoring) recommendations immediately [31], before producing the rest of the top-k list. Note that this is orthogonal to top-k view materialization [15, 36], since the requested result either matches exactly a cached one or not.

In this paper we develop scalable algorithms for GIR computation. We determine the conditions under which changes in the query vector invalidate the result, represent them in computational geometric terms, and make crucial observations that enable fast processing. Along the way, our GIR algorithms also compute the new top-k result should the query vector shift to any point on the GIR boundary. We verify the generality and efficiency of our methods using real and synthetic data with different characteristics.

2. RELATED WORK

As the notion of GIR builds on the top-k query, we first review top-k processing. Next, we survey safe regions for spatial queries, per-dimension (local) immutable regions for top-k queries, and sensitivity analysis in operations research. We also briefly cover convex hull computation, a foundation for our algorithms.

Given a database D and a scoring function, a top-k query retrieves the k records from D that achieve the highest scores. Top-k queries have been studied in various domains, including relational databases [21], middle-ware information systems [17], joins [33], and dynamic databases [1, 22]. Suiting our problem setting, BRS [32] is a top-k algorithm for low-dimensional data indexed by a spatial access method (e.g., an R-tree [19]). Designed for the broad class of monotone scoring functions, BRS applies the *branch-and-bound* methodology. Specifically, it uses a max-heap to organize the entries of visited R-tree nodes so as to access them in decreasing order of their *maxscore*. The *maxscore* of an R-tree node is the largest among the scores of its MBB corners (MBB stands for the node's minimum bounding box) and serves as an upper bound for the score of any record under the node. When a leaf node is accessed, the score of records inside is computed and the interim top-k result is updated accordingly. BRS terminates when the record with the k-th largest score in the interim result has a score no smaller than the *maxscore* of the last R-tree entry popped from the search heap. BRS is I/O optimal, meaning that it reads the minimum possible number of pages (R-tree nodes) from the disk.

Another related problem is the reverse top-k query [35, 34], which involves a database D and a collection of user preference functions represented as query vectors. A reverse top-k query returns those query vectors from the collection that include a given record $p \in D$ in their top-k result.

The most relevant existing study is [24], which determines immutable regions on individual decision factors. An immutable region there takes the form of a validity interval for an isolated query weight, *assuming that all the other weights are kept constant*. One interval is defined for each decision factor. We term those *local immutable regions* (LIRs) to distinguish from the GIR. In the context of Figure 1, [24] produces the same original marks and inner/outer polygons. However, due to the local nature of the LIRs, it cannot support simultaneous readjustments to multiple weights. More importantly, if a weight w_i is updated, the immutable regions for all the other factors are invalidated, *even if the new value of w_i remains within its LIR*. Referring to Figure 1 again, if w_3 shifts to 40 (which is still inside its permissible range) the technique in [24] needs to compute from scratch new LIRs for all the remaining factors. At the heart of LIR computation lie a pruning and a thresholding technique, both of which are tailored to LIRs and are inapplicable to GIR formation. Note that we may trivially derive LIRs from the GIR (as we discuss in Section 7.3), but the reverse does not hold.

Another related work is [30] which considers uncertain scoring functions and proposes methods to compute representative top-k results. It also introduces two sensitivity measures, STB and LIK. Given a top-k query with a linear scoring function, STB computes the largest ball around the query vector (in query space) where the top-k result remains the same. This ball is enclosed in (i.e., a subset of) our GIR, the latter being the *maximal* locus that preserves the result. Moreover, STB requires a scan of the dataset, which is prohibitive for large disk-resident data.

LIK defines a sensitivity measure that is equivalent to the ratio of the GIR volume to the volume of the query space. Other than the definition, however, [30] is not concerned with efficiency. It sketches an approach based on half-space[1] intersection that scans the entire dataset. With a time complexity of $O(n^{2^{d-2}})$, it is impractical for sizable databases. In Section 3.3 we sketch a straightforward approach to compute the GIR which, although it has a superior complexity of $O(n^{d/2})$ (compared to LIK), it is still hugely impractical, thus motivating the elaborate techniques in this paper.

In location-based services, while processing spatial queries such as nearest neighbors and window queries, servers face the problem of frequent index maintenance and result re-computation as data objects move around and update their locations. Safe region techniques are designed to alleviate this problem by assigning to each mobile object an area, known as *safe region*, within which it is guaranteed not to alter the result of any spatial query in the system [27, 25]. Safe region techniques are inapplicable to our problem, since they consider the notion of spatial proximity, as opposed to the score-based ranking involved in top-k processing.

Another topic that is related to GIR is sensitivity analysis in operations research, which addresses how changes in the input parameters affect the output of a model [29, 20]. This includes studying to what extent the input is allowed to change, so that the output continues to be optimal. There are two approaches. The first, one-factor-at-a-time (OFAT), varies one input factor while fixing the rest. Although simple to use, OFAT is known to miss out optimal combinations, as it cannot capture the interaction among multiple input parameters. This motivates the second approach, named multi-parameter sensitivity analysis (MPSA). MPSA varies multiple input factors simultaneously and observes the changes (in terms of variance) in the model output under the combined influence of the inputs. OFAT is also known as local sensitivity analysis, and MPSA as global sensitivity analysis. In a sense, the LIRs in [24] are analogous to OFAT, while our GIR to MSPA. Both the OFAT and MPSA techniques, however, consider the effect of different inputs to the variance of a value (i.e., the model's output). In our

[1]A half-space is either of the two parts into which a hyperplane divides a coordinate space. In two dimensions, it is a half-plane.

problem, instead, the output changes refer to updates in the order or composition of a top-k result.

Our solutions touch upon and employ convex hull computation algorithms. The convex hull of a dataset D in a d-dimensional space is the smallest convex set that encloses all the records in D. In two dimensions, the hull is a convex polygon, whereas in higher dimensions it is a convex polytope [6]. The boundary of the hull is represented by vertices (i.e., data records) and facets. Algorithms for efficient convex hull computation in two and three dimensions include gift-wrapping [10], Graham's scan [18], and Chan's algorithm [9]. For higher dimensions, Quickhull [3] and Clarkson's algorithm [14] are the most common, with a time complexity of $O(n^{d/2})$. Our solutions share some of the key operations in Clarkson's algorithm. Its crux is to incrementally build the hull by processing the data records one by one. If the current record lies above one or more facets of the hull (i.e., it is not enclosed by the hull), these facets are replaced by new ones that include the new record.

3. PRELIMINARIES

3.1 Definition of Global Immutable Region

We consider top-k processing in a low-dimensional space. The dataset D consists of n records. Each record $p \in D$ has an identifier and d numeric attributes $x_1, x_2, ..., x_d$ (also referred to as *dimensions*). The top-k query is defined by a vector of weights $q = (w_1, w_2, ..., w_d)$, called the *query vector*, which can be seen as a d-dimensional point. For ease of presentation, we assume that the data space and query space are normalized, i.e., data attributes and query weights have values in the range $[0, 1]$. The score of a data record p with respect to q is given by the dot product $\mathcal{S}(p, q) = q \cdot p = \sum_{i=1}^{d} w_i x_i$. This definition of $\mathcal{S}(p, q)$ is equivalent to what is also commonly referred to as a linear scoring function. The result R of the query is a list of k records with the highest scores in D, sorted in decreasing score order. In other words, $R = \{p_1, p_2, ..., p_k\}$ where p_i is the record with the i-th highest score ($1 \le i \le k$).

When the query vector q changes, we say that R is *preserved* if both its composition and the score order among its members are unaltered. The problem we address in this paper is the computation of the maximal locus in the query space where R is preserved. We call this locus the *global immutable region* (GIR) of q.

DEFINITION 1. ***Global immutable region (GIR)****. Given a dataset D and a top-k query q with result $R = \{p_1, p_2, ..., p_k\}$, the GIR is the locus of all vectors q' in query space where*

1. *$\mathcal{S}(p_i, q') \ge \mathcal{S}(p_{i+1}, q')$ for each $i \in [1, k)$, and*
2. *$\mathcal{S}(p_k, q') \ge \mathcal{S}(p, q')$ for every record $p \in D \backslash R$.*

3.2 Nature of Global Immutable Region

A first step to understanding the problem is to determine the shape of the GIR. For the score order within result R to be preserved, $k - 1$ conditions must hold, each of the form $\mathcal{S}(p_i, q') \ge \mathcal{S}(p_{i+1}, q')$ (for $i \in [1, k)$). For the k-th result record p_k to remain ahead of all the non-result records p, another $n - k$ conditions must hold, each of the form $\mathcal{S}(p_k, q') \ge \mathcal{S}(p, q')$. Due to the form of the scoring function, each of these $n - 1$ conditions corresponds to a half-space in query space, whose defining hyperplane passes through the origin[2]. The top-k result is preserved if and only if the query vector remains within the intersection of these $n - 1$ half-spaces. This intersection constitutes the GIR of the query. Any intersection of half-spaces (and therefore the GIR too) is by definition a convex polytope.

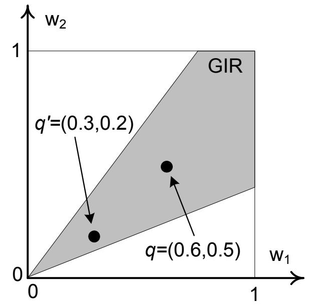


Figure 2: GIR example in 2-dimensional (query) space

Figure 2 shows how the GIR looks in 2-dimensional space (i.e., $d = 2$). Query vector $q = (0.6, 0.5)$ represents the user's original weight setting. In two dimensions, each of the $n - 1$ conditions derived from Definition 1 corresponds to a half-plane (instead of a half-space), whose defining line (instead of defining hyperplane) passes through the origin. Their intersection (i.e., the GIR) is a wedge like the one shown shaded in the figure. Any query vector lying inside this area (like the depicted $q' = (0.3, 0.2)$) is guaranteed to preserve the top-k result. Furthermore, this is the maximal locus in the query space where result R is preserved.

Without loss of generality, each line that bounds the GIR corresponds to one of the original $n - 1$ conditions. This implicitly determines what the new top-k result will be if the query shifts to a particular line. For instance, assume that the upper bounding line of the GIR corresponds to condition $\mathcal{S}(p_k, q') \ge \mathcal{S}(p, q')$, where p is a non-result record. If the query is adjusted to fall on this line, the new result will be the same as the current one, except that p will replace the k-th record in R. If the bounding line corresponds to condition $\mathcal{S}(p_i, q') \ge \mathcal{S}(p_{i+1}, q')$, the resulting update in R is that p_{i+1} overtakes record p_i in score, i.e., the perturbation is a reordering between p_i and p_{i+1} in the result.

In the following we focus on the GIR computation. *Determining the exact result perturbation when the query moves to the boundary of the GIR happens along the way, by identifying the record responsible for each of the half-spaces that bound the GIR.*

3.3 Challenges, Assumptions and Setting

Our goal is to develop efficient GIR computation algorithms that are scalable to large datasets. The discussion in the previous session hints at a possible GIR computation approach, based on half-space intersection. However, deriving the $n - 1$ half-spaces stated in Definition 1 requires scanning the entire dataset, and incurs a large data access cost. Furthermore, performing the intersection of $n - 1$ half-spaces requires an excessive amount of computations, specifically, $\Omega(n^{d/2})$ which explodes for large datasets. The main challenge we address is how to reduce the number of records/half-spaces considered so as to minimize (i) the data access cost to retrieve those records, and (ii) the processing time for half-space intersection, while still guaranteeing correct/exact GIR computation.

Targeted at large scale, low-dimensional datasets, we assume that D is indexed by a spatial access method. Our implementation

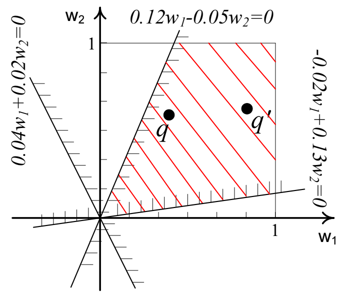
[2]Consider for example condition $\mathcal{S}(p_k, q') \ge \mathcal{S}(p, q')$, which in dot product notation is $p_k \cdot q' \ge p \cdot q' \Rightarrow (p_k - p) \cdot q' \ge 0$. Since vector $(p_k - p)$ is fixed (with attribute values in both records being constant), the inequality implies that q' lies on a half-space that is bounded by hyperplane $(p_k - p) \cdot q' = 0$. The latter passes through the origin of the query space.

Symbol	Description
d	Data dimensionality
D	Dataset in $[0,1]^d$
n	Number of records in D
p	A data record in D
q	User query (vector in $[0,1]^d$)
$\mathcal{S}(p,q)$	Score of p w.r.t. q
R	Top-k result
p_i	The i-th record in R ($1 \le i \le k$)
$\mathcal{SL}$	Skyline of $D \backslash R$
$\mathcal{CH}$	Convex hull of non-result records
$\mathcal{CH}'$	Convex hull of set $\{p_k\} \cup D \backslash R$

Table 1: Notation

	x_1	x_2	$p \cdot q$
p_1	.54	.5	.516
p_2	.5	.48	.488
p_3	.52	.35	.418
p_4	.4	.4	.4

(a) Top-k records



(b) Half-planes in query space

Figure 3: Phase 1 example ($k = 4$, $d = 2$)

employs the ubiquitous R*-tree [4], though our techniques apply directly to other space- or data-partitioning indices. The index and data could reside in memory or on disk, the latter being our default setting (although we evaluate our techniques in both scenarios).

When a query q is posed, prior to GIR computation, its top-k result R must be retrieved. For this we employ the state-of-the-art BRS technique [32], yet our work is not directly dependent on the choice of top-k algorithm. To facilitate subsequent GIR-related processing, we maintain all the data records encountered by BRS (but not included in the top-k result), as well as its search heap.

After the top-k result is produced, GIR computation commences. That comprises two phases. The first derives an interim GIR based on the first set of conditions in Definition 1 (considering only records in R). The second phase shrinks the interim GIR according to the second set of conditions (imposed by non-result records). In Table 1 we summarize the notation used in the paper.

4. PROCESSING IN PHASE 1

Given the original top-k result R, the first phase derives an interim GIR from the first set of conditions in Definition 1. There are $k-1$ conditions, each defining a half-space in query space. The interim GIR in Phase 1 is obtained by intersecting these half-spaces. Formally, the interim GIR is the following polytope in query space:

$$\bigcap_{i=1}^{i=k-1} \{q' | q' \in [0,1]^d \text{ such that } (p_i - p_{i+1}) \cdot q' \ge 0\} \quad (1)$$

Consider a query with $q = (0.4, 0.6)$ and $k = 4$. Suppose that a top-k algorithm (BRS in our implementation) reports in the result records p_1, p_2, p_3, and p_4, whose attributes and scores are shown in Figure 3(a). Phase 1 commences by deriving the half-plane that preserves the order between p_1 and p_2, i.e., half-plane $(p_1 - p_2) \cdot q' \ge 0 \Rightarrow 0.04w_1 + 0.02w_2 \ge 0$. Figure 3(b) represents the half-plane by its bounding line ($0.04w_1 + 0.02w_2 = 0$) in the

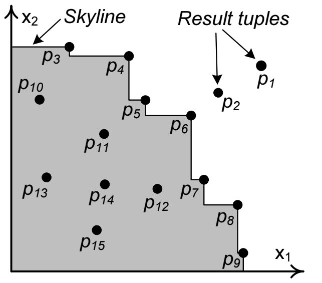


Figure 4: SP example ($k = 2, d = 2$, data space)

query space. Similarly, preserving the order between p_2 and p_3 defines the half-plane $(p_2 - p_3) \cdot q' \ge 0 \Rightarrow -0.02w_1 + 0.13w_2 \ge 0$. In turn, p_3 and p_4 define $(p_3 - p_4) \cdot q' \ge 0 \Rightarrow 0.12w_1 - 0.05w_2 \ge 0$. The interim GIR is the striped area at the intersection of the three half-planes. Any query vector in this area is guaranteed to uphold the score order among the four result records. We use a 2-dimensional example here for ease of presentation. The process is very similar in higher dimensions.

Phase 1 is fast because the number of result records k is typically much smaller than the number of non-result records that need to be considered. It is also uniform across all methods in our framework. The distinction among them lies entirely in Phase 2, the bottleneck in GIR computation.

5. BASIC METHODS FOR PHASE 2

Phase 2 further shrinks the GIR to ensure that no non-result record can overtake the k-th result record p_k in score. As explained in Section 3, the efficiency of this phase hinges on the ability to reduce the number of non-result records examined. In this section, we present two basic methods for pruning the set $D \backslash R$ to safely discard records that could not affect the GIR.

5.1 Skyline Pruning Method

The first method relies on the *skyline* operator [8]. The skyline of a set includes only those members that are not *dominated* by any other member. In our context, we say that record p dominates another record p' if the value of p in every dimension is no smaller than the corresponding value of p', and the two records differ on at least one dimension. Due to the definition of dominance, record p could have a score no smaller than p' under any monotone scoring function [21] (which is a superclass of the linear scoring functions we assume). Since the score of p' never exceeds that of p regardless of the query vector, record p' cannot overtake the current p_k before p overtakes p_k. In other words, $\mathcal{S}(p, q') \ge \mathcal{S}(p', q')$ for any $q' \in [0,1]^d$ so, by satisfying the condition $\mathcal{S}(p_k, q') \ge \mathcal{S}(p, q')$, our GIR automatically also upholds $\mathcal{S}(p_k, q') \ge \mathcal{S}(p', q')$. Hence, it is safe to ignore p' in GIR computation.

To generalize, we may safely prune the set $D \backslash R$ by retaining only the records in its skyline $\mathcal{SL}$. The final GIR can be derived by intersecting the interim GIR from Phase 1 with the half-spaces formed from inequalities $(p_k - p) \cdot q' \ge 0$ for each record $p \in \mathcal{SL}$. We term this approach *Skyline Pruning* (SP). Figure 4 shows a 2-dimensional example where $k = 2$, the dataset comprises records $p_1, p_2, ..., p_{15}$, and the result includes p_1 and p_2. The skyline $\mathcal{SL}$ of the non-result records includes $p_3, p_4, ..., p_9$, which are the only records considered by SP in Phase 2. Records that fall in the shaded area are dominated by at least one member of the skyline.

Although SP can potentially disqualify many records from $D \backslash R$, the cardinality of the skyline may still be large. As an indication, the number of records in $\mathcal{SL}$ is in the order of $O((\log n)^{d-1})$ [5]

for independent data, while there are common (e.g., anti-correlated) distributions where the cardinality is even higher.

In terms of implementation, the state-of-the-art algorithm for skyline computation on spatial access methods is BBS [26], which follows the *branch-and-bound* paradigm. It utilizes an R-tree on the dataset to incrementally retrieve nearest neighbors (NNs) to the top corner of the data space, i.e., to point $(1, 1, ..., 1)$. The first NN is guaranteed to be in the skyline, and is used to prune the part of the space it dominates. Then, the next NN is retrieved in the remaining part of the space; it is also included in the skyline and used to further prune the search space. The process continues until no more NNs can be found in the non-dominated part of the space.

To see how BBS applies in our situation, recall that (before GIR computation) the top-k result was produced by BRS, and that we have retained its search heap and all the non-result records that were encountered during its execution. We initialize $\mathcal{SL}$ by computing the skyline of the encountered non-result records (using any main memory algorithm [8]), and then invoke BBS on the retained search heap to consider records that may belong to $\mathcal{SL}$ but were not accessed during BRS execution. Recall that the search heap of BRS is organized on *maxscore*. That is, in our BBS execution, instead of incrementally retrieving NNs to the top corner of the data space, we retrieve records in decreasing $\mathcal{S}(p, q)$ order. This does not affect the correctness of BBS, since the distance from the top corner of the data space (in vanilla BBS) can be replaced by any monotone scoring function to determine the retrieval order [26]. Another modification is that any record p retrieved by BBS is inserted into $\mathcal{SL}$ only if it is not dominated by any of its members, while if p dominates any existing members, the latter are removed from $\mathcal{SL}$.

5.2 Convex Hull Pruning Method

Our second basic solution for Phase 2 prunes non-result records based on the concept of the convex hull. If we treat the records of a dataset as points in d-dimensional space and compute their convex hull, the geometric properties of the hull guarantee that the top-1 record under any linear scoring function (defined over the same d dimensions) lies on the hull[3] [23, 11].

Let $\mathcal{CH}$ be the convex hull of the non-result records. The above property guarantees that for any query vector $q' \in [0, 1]^d$ and any record p' that is strictly enclosed by $\mathcal{CH}$ (as opposed to lying on it), there is at least one record p on the hull (i.e., $p \in \mathcal{CH}$) such that $\mathcal{S}(p, q') \geq \mathcal{S}(p', q')$. This implies that p' cannot overtake the current k-th result record p_k until some record on the hull has overtaken p_k. Hence, only records on $\mathcal{CH}$ could affect the GIR.

Utilizing the above observation directly would prune $D \backslash R$ by retaining only those records that fall on its convex hull. Nevertheless, we can do better. To exemplify, Figure 5 illustrates the convex hull of the non-result records, assuming the same setting/dataset as Figure 4. The records that lie on (as opposed to inside) $\mathcal{CH}$ are $p_3, p_4, p_6, p_8, p_9, p_{15}, p_{13}, p_{10}$. However, we observe that p_{15}, p_{13}, p_{10} are dominated by p_4, and could therefore not affect the GIR (by the same reasoning as in SP). This observation gives rise to our second baseline algorithm, *Convex Hull Pruning* (CP), which considers in Phase 2 only those non-result records that belong to the skyline and at the same time fall on the convex hull of $D \backslash R$, i.e., records $p \in \mathcal{SL} \cap \mathcal{CH}$. Referring to the example in Figure 5, CP considers only records p_3, p_4, p_6, p_8, p_9.

In implementing CP, we first retrieve the skyline of $D \backslash R$, using the same BBS-based approach as SP. Following that, we compute the convex hull *of the skyline records only* (using Clarkson's algorithm [14]). This hull is shown shaded in the example of Figure 5.

[3]When we say that a record "lies on" or "belongs to" the hull, we mean that it lies *on the boundary* of the hull.

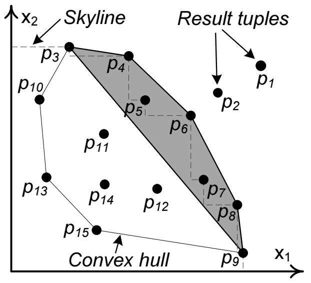
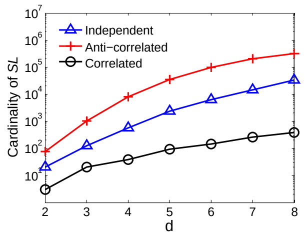
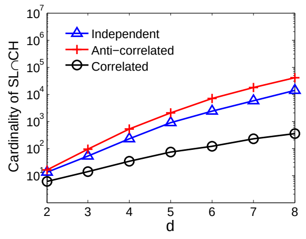


Figure 5: CP example ($k = 2, d = 2$, data space)



(a) Cardinality of $\mathcal{SL}$



(b) Cardinality of $\mathcal{SL} \cap \mathcal{CH}$

Figure 6: SP and CP effectiveness (n = 1M, k = 20)

The records that lie on the hull are used for half-space intersection with the interim GIR from Phase 1 to derive the final GIR. An alternative approach would be to compute the convex hull before disqualifying the dominated among its records. This would be inefficient, because it would access parts of the space that are too far (and irrelevant) from the GIR, like the vicinity of p_{15}, p_{13}, p_{10} in Figure 5. Moreover, the only existing convex hull algorithm that utilizes a spatial index [7] applies only to 2-dimensional space.

5.3 Performance Indications

We now provide preliminary indications of the performance (and shortcomings) of SP and CP. We use three types of synthetic datasets (independent, anti-correlated, and correlated) with cardinality 1M each. We defer the description of these data, but note that they are standard benchmarks for preference-based queries.

Figure 6(a) plots, for different dimensionalities, the number of records that belong to the skyline of $D \backslash R$, i.e., the non-result records that SP needs to process in Phase 2. As anticipated, although SP prunes a large fraction of $D \backslash R$, it still needs to consider numerous records. The problem is exacerbated with growing d.

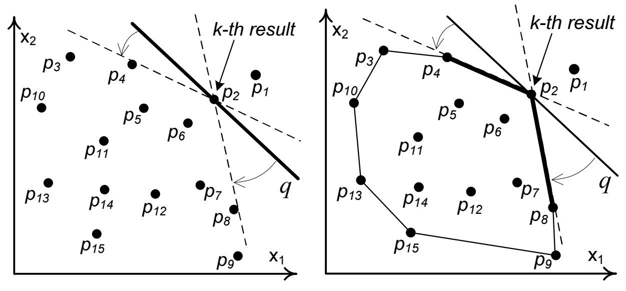
CP retains only the records in $\mathcal{SL} \cap \mathcal{CH}$, i.e., a subset of those examined in SP. In Figure 6(b) we present the number of records remaining after CP pruning in the same setting as Figure 6(a). CP aggressively reduces the number of non-result records in Phase 2. However, its effective pruning comes at the price of a convex hull computation over $\mathcal{SL}$, which entails substantial processing time.

6. ADVANCED SOLUTION FOR PHASE 2

The shortcomings of SP and CP motivate the development of an efficient and scalable Phase 2 algorithm that also copes better with dimensionality. We call this method *Facet Pruning* (FP). To provide the intuition behind it, we explore the nature of top-k query.

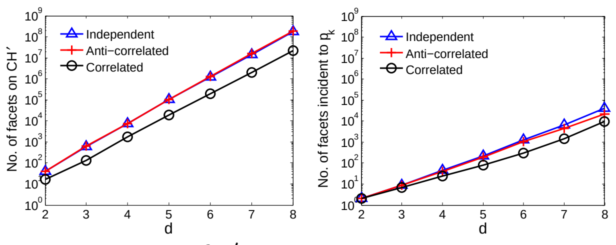
6.1 Rationale of Facet Pruning Method

In Figure 7(a) we assume the same dataset and top-k query as in Figures 4 and 5. Computing the top-k result can be seen as scanning the data space from its top corner (i.e., point $(1, 1)$) towards



(a) Tilting around p_k (b) Determining facets

Figure 7: Intuition behind FP ($k = 2, d = 2$, data space)



(a) Facets on $\mathcal{CH}'$ (b) Facets incident to p_k

Figure 8: FP effectiveness (n = 1M, $k = 20$)

the origin, with a line (or hyperplane, in higher dimensions). The orientation of the line is fixed, and it is determined by the query vector[4]. The first encountered record has the highest score, the second encountered record has the next highest score, and so on. The search stops after finding k data records, which form the result R.

In our example, the sweeping line would first encounter p_1 and stop when it hits p_2 (recall that $k = 2$). The line position when it hits $p_k \equiv p_2$ is defined by equation $w_1x_1 + w_2x_2 = \mathcal{S}(p_k, q)$ and is drawn in bold in Figure 7(a). It partitions the data space into two regions – all records above the line have higher scores than p_k and belong to R, whereas records below the line have lower scores (than p_k) and are not among the top-k. Any records on the line have the same score as p_k (yet, without loss of generality, we assume that there are no ties).

For now, let us ignore reorderings within R, and focus on the second set of conditions in Definition 1, which ensure that no non-result record overtakes p_k in score. Returning to Figure 7(a) and the nature of top-k processing, a tilt in the orientation of the sweeping line is equivalent to a shift in the query vector. Assume that we pin the sweeping line at p_k but allow it to rotate. A rotated position of the line is *permissible* if it keeps all non-result records below it. This implies that p_k still scores higher than them, and the query vector q' that corresponds to the new line orientation preserves R.

Consider a clockwise rotation in Figure 7(a). The first record hit by the line (i.e., p_8) bounds the permissible clockwise rotations, because any further tilting would perturb the result (p_8 would score higher than the current $p_k \equiv p_2$). Similarly, the permissible anticlockwise rotations are bounded by p_4. Any other non-result record cannot provide a stricter rotation bound than p_4 and p_8.

We make a crucial observation that hints at a general methodology to identify records like p_4 and p_8. In Figure 7(b) we show the convex hull of set $\{p_k\} \cup D\backslash R$ (i.e., the set of non-result records extended by p_k). The two facets that are incident[5] to p_k on the hull (i.e., facets $\overline{p_4, p_2}$ and $\overline{p_2, p_8}$) correspond to the records of interest (p_4 and p_8, respectively).

This is aligned with the property of the convex hull that each of its facets keeps all the records on one of its sides (specifically, the one toward the interior of the hull), which holds for any dimensionality [6]. Since the sweeping line (or the sweeping hyperplane, in higher dimensions) is pinned at p_k, its permissible orientations are determined only by *the facets of the convex hull that are incident to* p_k. We call the records that are incident to those facets *critical*. They are the only non-result records that could affect the GIR.

[4]Vector q is a *normal vector* to the sweeping line (or hyperplane, for $d > 2$), meaning that it is perpendicular to the line (or hyperplane, respectively).

[5]A record and a facet are incident to each other if the record lies at a corner of the facet. In two dimensions, for example, a facet is a line segment, and it is incident to the two records at its endpoints.

Intuitive as this fact may sound, translating it into an efficient Phase 2 algorithm is challenging. Let $\mathcal{CH}'$ denote the convex hull of set $\{p_k\} \cup D\backslash R$. A naïve implementation would compute $\mathcal{CH}'$, get the hull facets that are incident to p_k, collect their incident records (i.e., the critical records), and derive the GIR using only these records for half-space intersection. Convex hull computation, however, has a time complexity of $\Omega(n^{d/2})$, which is equivalent to the complexity of the exhaustive half-space intersection described in Section 3.3 (actually, convex hull computation and half-space intersection are dual to each other [6]). The situation becomes worse if one considers that there is no off-the-shelf convex hull algorithm for disk-resident data in more than two dimensions.

To alleviate the problem, our FP approach computes *only the relevant part of the convex hull*, i.e., only the hull facets that are incident to p_k. This approach provides scalability with respect to both dataset cardinality and dimensionality. We thoroughly demonstrate this fact in Section 8, yet here we provide some preliminary empirical evidence that substantiates the FP rationale. In Figures 8(a) and 8(b) we plot the total number of facets in $\mathcal{CH}'$ and the number of facets that are incident to p_k, for the same setting as Figure 6. The charts suggest that FP needs to compute/consider only a very small fraction of the hull facets. We remark that a critical record may be incident to more than one facet, i.e., the number of critical records may be smaller than the number of facets shown in Figure 8(b). For example, for independent data and $d = 4$, there are 45 facets incident to p_k and 16 critical records, while for $d = 6$ the number of incident facets is 1258 and that of critical records is 98.

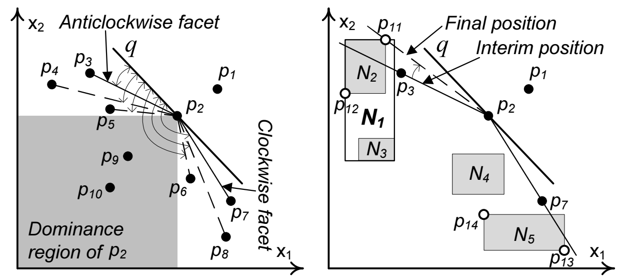
Next, we present the detailed FP algorithm. We distinguish between the FP versions for $d = 2$ (discussed in Section 6.2) and $d > 2$ (covered in Section 6.3), because the nature of 2-dimensional space allows for special-purpose enhancements.

6.2 Facet Pruning in Two Dimensions

The convex hull/incident facet observation exemplified in Figure 7(b) is general and particularly useful for $d > 2$. In the special case of 2-dimensional space, however, the visualization in Figure 7(a) already suggests an effective processing methodology. That is, FP needs to simply identify the two records that constrain the rotation of the sweeping line around p_k in the clockwise and anticlockwise directions. Recall that some of the records in $D\backslash R$ have already been fetched from the disk by BRS during the initial top-k computation, and kept in memory, as explained in Section 3.3. Let T denote this set of records. The remaining non-result records are on the disk and are accessible via the R-tree on D.

FP consists of two steps. The first considers T and identifies two *candidate* critical records or, equivalently, *interim* facets incident to p_k. The second refines those facets by exploring data from the disk (using the index), until it identifies the two actual critical records.

The first step starts by removing from T the records that are dominated by p_k, as they cannot overtake it in score under any query vector $q' \in [0, 1]^2$. Then, it angularly sorts the remaining records



(a) First step (b) Second step

Figure 9: FP example ($k = 2, d = 2$, data space)

in T. That is, for every record $p \in T$, it computes the angle by which the sweeping line must rotate (in the anticlockwise direction) in order to hit p. It then picks as candidate critical records the two with the minimum and maximum angles. The minimum-angle record corresponds to the anticlockwise interim facet, and the maximum-angle record to the clockwise interim facet.

We demonstrate the first step of FP in Figure 9(a), where T includes records $p_3, p_4, ..., p_{10}$. We first remove p_9 and p_{10} from T because they are dominated by $p_k \equiv p_2$. The angle for each of the remaining records in T is illustrated by a curved two-headed arrow. The minimum-angle record is p_3, while p_7 is the maximum-angle one. These records define the interim facets $\overline{p_3, p_2}$ and $\overline{p_2, p_7}$. Note that if the area above the dominance region of p_2 was empty (i.e., if p_3, p_4, p_5 were not there), the anticlockwise interim facet would be the line segment between p_2 and its projection on the vertical axis. Similarly, if p_6, p_7, p_8 were not there, the clockwise interim facet would connect p_2 with its projection on the horizontal axis.

In the second step of FP we consider non-result records that were not encountered during top-k computation (i.e., records not fetched from the disk as yet), utilizing the R-tree on D and the retained search heap of BRS. We iteratively pop the heap, using its *maxscore* key as is. If the popped entry corresponds to an R-tree node, and the minimum bounding box (MBB) of the node lies completely below the interim facets, we prune/ignore it. Alternatively (i.e., if at least a part of the MBB is above an interim facet), we fetch the node from the disk. If it contains index entries (i.e., it is an internal node of the R-tree), we push its children into the heap, with key equal to their *maxscore* according to q. On the other hand, if it contains data records (i.e., it is a leaf of the R-tree), we consider each of these records p as follows. If p lies above an interim facet, we update the facet to connect p_k with p; otherwise, we ignore p. The process terminates when the heap becomes empty, reporting the interim facets as the final ones. Our implementation uses the beneath-and-beyond technique in [3] to check whether an MBB or record lies below the interim facets.

In Figure 9(b) we demonstrate the second step of FP, continuing the example of Figure 9(a). The interim facets $\overline{p_3, p_2}$ and $\overline{p_2, p_7}$ from the first step are shown as solid lines. Records p_1, p_2, p_3, and p_7 that are already known to the algorithm, are represented as solid points. Records shown as hollow points are on the disk and have not been encountered yet. In the beginning of the process, the search heap of BRS is assumed to include index entries that correspond to R-tree nodes N_5, N_4, N_1 (stated here in decreasing order of *maxscore*). The first entry popped from the heap corresponds to N_5. A part of its MBB lies above the clockwise facet $\overline{p_2, p_7}$. Hence, node N_5 is fetched from the disk. It includes two records, p_{13} and p_{14}, none of which lies above $\overline{p_2, p_7}$. Therefore, the clockwise facet remains as is. The second popped entry corresponds to N_4, which lies completely below both interim facets and is thus ignored.

The next popped entry corresponds to N_1. The node is read from the disk, since part of its MBB is above the anticlockwise interim facet $\overline{p_3, p_2}$. The two child entries in N_1 point at N_2 and N_3, which are pushed into the heap (with key equal to their *maxscore* according to q). The heap is popped again, producing the entry of N_2. Node N_2 is fetched from the disk, because it is not completely below facet $\overline{p_3, p_2}$. Between records p_{11}, p_{12} contained in the node, p_{11} lies above the anticlockwise facet. The facet is therefore updated to $\overline{p_{11}, p_2}$, shown as a dashed line in the figure. The last entry popped from the heap corresponds to N_3, which is below both the current facets and hence ignored. The heap is now empty and the final facets derived are $\overline{p_{11}, p_2}$ and $\overline{p_2, p_7}$. The correctness of the second step relies on the fact that (i) any pruned index entry or data record lies below both interim facets and therefore is unable to update either of them, and (ii) the process terminates only when the search heap becomes empty.

The facets derived by FP indicate the critical records, e.g., in Figure 9 the critical records are p_{11} and p_7. From these records, we derive half-planes $(p_2 - p_{11}) \cdot q' \geq 0$ and $(p_2 - p_7) \cdot q' \geq 0$, respectively, that embody the second set of conditions in Definition 1. The GIR from Phase 1 is intersected with those two half-planes to produce the final GIR. We summarize FP in Algorithm 1. Lines 1-2 implement the first step of FP to compute the interim anticlockwise and clockwise facets f_a and f_c, respectively. Lines 3-16 correspond to the second step of FP, while Lines 17-19 intersect the interim GIR from Phase 1 with the half-planes derived from critical records p_a, p_c to produce the final GIR.

Algorithm 1: Facet Pruning

```
   Input: heap $H$ of BRS, query vector $q$, $k$-th result $p_k$
   Output: Final GIR
1  $T \leftarrow$ set of non-result records encountered by BRS;
2  $f_a, f_c \leftarrow InterimFacets(T, q, p_k)$;
3  while $H$ is not empty do
4      Pop the top entry $e$ from $H$;
5      if $e$ is below both facets $f_a, f_c$ then
6          Continue;
7      if $e$ corresponds to a leaf node then
8          Fetch the node from the disk;
9          for each record $p$ in the node do
10             if $p$ is above $f_a$ then
11                 Update $f_a$ to $\overline{p, p_k}$;
12             if $p$ is above $f_c$ then
13                 Update $f_c$ to $\overline{p_k, p}$;
14     if $e$ corresponds to an internal node then
15         Fetch the node from the disk;
16         Push all the child entries of the node into $H$;
17 $p_a, p_c \leftarrow$ records incident to $f_a$ and $f_c$, respectively;
18 Intersect GIR from Phase 1 with half-planes
   $(p_k - p_a) \cdot q' \geq 0$ and $(p_k - p_c) \cdot q' \geq 0$;
19 Return GIR;
```

6.3 Facet Pruning in Higher Dimensions

In more than two dimensions, a top-k query q can be seen as a sweeping hyperplane (to which the query vector is perpendicular). Following the FP paradigm, if we pin the sweeping hyperplane at p_k and allow it to rotate freely in any direction, the critical records are those (among the non-result records) that bound its movement, so as to keep all non-result records under the hyperplane. To vi-

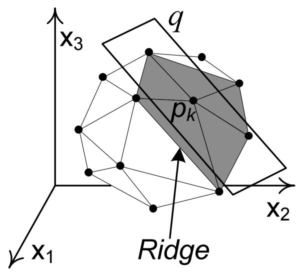


Figure 10: Facets incident to p_k ($d = 3$, data space)

sualize, we use a 3-dimensional example in Figure 10. The figure illustrates the convex hull $\mathcal{CH}'$ of set $\{p_k\} \cup D \backslash R$ and the sweeping plane that is pinned at p_k. The goal in FP is to compute the hull facets that are incident to p_k so as to collect the critical records. In our example, there are five such facets (shown shaded) that lead to five incident/critical records, excluding p_k.

Before we present algorithmic details, we note that a facet is a $(d-1)$-dimensional object that is generally defined by d records. A *ridge* is a $(d-2)$-dimensional object representing the intersection of two neighboring facets. For $d = 3$, the ridge is a line segment where two facets meet, and it is defined by the two records at its endpoints. In Figure 10 we point out a ridge at the intersection of a shaded and a normal facet.

The main idea in FP is to avoid computing the entire hull by maintaining only the facets that are incident to p_k. This is carried out in two steps, at the heart of which lies a strategy that incrementally updates the set of incident facets as new records are considered. The first considers the set T of non-result records encountered by BRS, and the second those still on the disk.

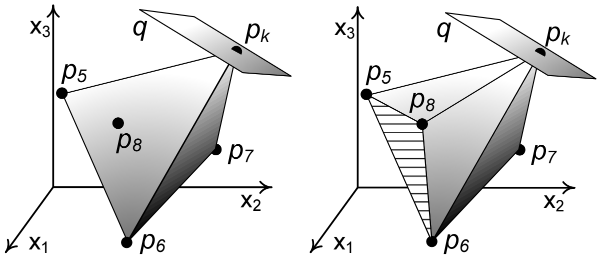
6.3.1 First Step of FP

The first step starts by discarding/removing from T those records that are dominated by p_k. Then, it draws d records[6] from T and computes a convex hull on these records and p_k (note that the cost to compute the convex hull of $d + 1$ records is trivial). The hull facets that are incident to p_k are maintained in set $\mathcal{F}$, while the rest are discarded. Consider the 3-dimensional example in Figure 11(a) where p_5, p_6, p_7 are drawn from T. From the convex hull of p_5, p_6, p_7, p_k, we keep in $\mathcal{F}$ the three facets incident to p_k, i.e., $(p_k, p_5, p_6), (p_k, p_6, p_7)$, and (p_k, p_7, p_5). The bottom facet (p_5, p_6, p_7) is not incident to p_k and is discarded.

Next, we process each record p in T and incrementally update $\mathcal{F}$. If p lies below all the facets in $\mathcal{F}$, it is discarded. Otherwise, we update $\mathcal{F}$ following a process reminiscent of Clarkson's algorithm, yet focused on facets incident to p_k only.

Specifically, we initialize a set $\mathcal{F}_v$ and place in it all the facets from $\mathcal{F}$ that p lies above of. In Figure 11(a) the only facet that is below p_8 is (p_k, p_5, p_6). Then, we collect all the ridges that are shared between a facet in $\mathcal{F}_v$ and a facet in $\mathcal{F} \backslash \mathcal{F}_v$. In the literature these are called *horizon ridges*. In our example the horizon ridges are $\overline{p_5, p_k}, \overline{p_6, p_k}$ and $\overline{p_5, p_6}$. Among them, we keep only those incident to p_k (i.e., the former two). We update $\mathcal{F}$ by (i) removing the facets that belong to $\mathcal{F}_v$, and (ii) inserting one new facet for each of the retained horizon ridges (the new facets are formed by connecting p with the respective ridge). Figure 11(b) shows the updated $\mathcal{F}$ after processing p_8. Observe that facet (p_k, p_5, p_6) is removed

[6]If T contains fewer than d records, we may use instead the projections of p_k on each of the d axes. This is equivalent to our strategy in Section 6.2 when the first step of FP encountered empty areas around the dominance region of p_k.



(a) Processing record p_8 (b) Updated set of facets $\mathcal{F}$

Figure 11: FP example ($d = 3$, data space)

and is replaced by two new facets. The first is defined by p_8 and horizon ridge $\overline{p_5, p_k}$, and the second by p_8 and $\overline{p_6, p_k}$. Note that the striped facet was never created nor inserted in the updated $\mathcal{F}$. That facet would not be incident to p_k, and we avoided its unnecessary formation through our strategy to discard those horizon ridges that were not incident to p_k (i.e., ridge $\overline{p_5, p_6}$).

We optimize the first step with a heuristic. Recall that in the beginning of this step, we draw d records from T to (build a convex hull and) form the initial set $\mathcal{F}$. Instead of a random choice, we pick the d records from T with the maximum values along each of the d dimensions. The rationale is that many non-result points are likely to lie below the formed facets, and thus be pruned directly later on.

6.3.2 Second Step of FP

In the second step we refine $\mathcal{F}$ by considering non-result records that have not been encountered (not fetched into memory) before. $\mathcal{F}$ is updated gradually as we pop entries from the search heap of BRS. The exploration of the R-tree is similar to the 2-dimensional case. Index nodes are pruned (ignored) if they lie completely below each facet in $\mathcal{F}$; otherwise, they are read from disk. When an internal node is read, its child entries are pushed into the heap with key set to their *maxscore* according to q. When a leaf node is read, each of its records is checked against $\mathcal{F}$, the latter being updated if the record lies above some of its facets. The update process is identical to the description in Section 6.3.1. The process terminates when the heap becomes empty, and the critical records are collected from the final set $\mathcal{F}$. Every critical record p is mapped into a half-space of the form $(p_k - p) \cdot q' \geq 0$ and intersected with the interim GIR from Phase 1 in order to produce the final GIR[7].

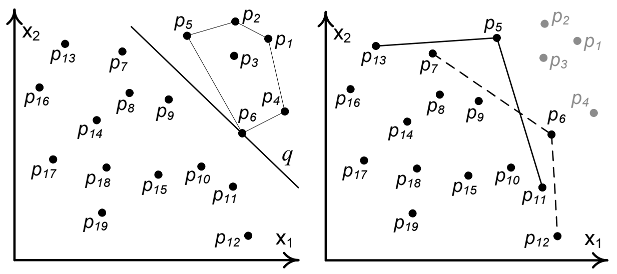
Although the visual examples used in Section 6.3 are for $d = 3$, both steps of the FP methodology apply to higher dimensions without modification, by simply using the conventional notions of facets and ridges in the respective space.

We conclude the discussion about FP with a note on its complexity. According to [12], the number of facets on $\mathcal{CH}'$ is $O(n^{d/2})$, while the number of facets incident to a record on the hull (e.g., to p_k) is $O(n^{\frac{d}{2}-1})$. Computing these facets is the bottleneck in FP. FP uses a process based on Clarkson's algorithm, whose complexity for the entire hull is $O(n^{d/2})$. Following a similar reduction to [12], the cost to compute only the facets incident to p_k is $O(n^{\frac{d}{2}-1})$.

7. EXTENSIONS AND VISUALIZATION

In this section we extend our methodology to order-insensitive GIR, discuss the handling of non-linear scoring functions, and describe possible GIR visualization techniques.

[7]An optimization is possible where the interim GIR from Phase 1 is mapped into facets and can be incorporated into the first step of Phase 2 to further "tighten" the criteria for fetching nodes from the disk in the second step of Phase 2.



(a) Result pruning (b) Facet sets $\mathcal{F}_5$ and $\mathcal{F}_6$

Figure 12: GIR* computation ($k = 6, d = 2$, data space)

7.1 Order-Insensitive GIR

A variant of the GIR problem arises when the user or application is concerned only about the composition of the top-k result (but not the order of records in it). The *order-insensitive* GIR is the maximal locus in query space where the composition of R is preserved. We denote it as GIR*.

DEFINITION 2. ***Order-insensitive GIR (GIR*).*** *Given a dataset D and a top-k query q with result $R = \{p_1, p_2, ..., p_k\}$, the GIR* is the locus of all vectors q' in query space where*

$$\mathcal{S}(p_i, q') \geq \mathcal{S}(p, q')$$

for each $i \in [1, k]$ and every record $p \in D \backslash R$.

The GIR* is defined by looser conditions than the (order-sensitive) GIR, and hence it fully encloses the latter. Definition 2 suggests a straightforward processing approach. Consider a result record $p_i \in R$. Let GIR$_i$ be the GIR derived if we (skip Phase 1 and) apply Phase 2 by having p_i play the role of p_k, using any of the methods in Sections 5 or 6. GIR$_i$ is the maximal region in query space where $\mathcal{S}(p_i, q') \geq \mathcal{S}(p, q')$ for every record $p \in D \backslash R$. By Definition 2, the GIR* is the intersection of the GIR$_i$ regions for each $i \in [1, k]$, i.e., GIR$^* = \bigcap_{i=1}^{i=k}$ GIR$_i$.

To optimize this process, we observe that not every record in R could affect the GIR*. In Figure 12(a), where $k = 6$ and $R = \{p_1, ..., p_6\}$, we consider the convex hull of R. Any record p_j inside the hull (e.g., p_3) can be ignored. The rationale is similar to CP in Section 5.2. For every query vector q' there is at least one result record p_i that lies on the hull, such that $\mathcal{S}(p_i, q') \leq \mathcal{S}(p_j, q')$. Thus, for any non-result record p to overtake p_j in score, p would first have to overtake a result record that lies on the hull.

Further result pruning is possible. Observe that p_2 dominates p_5, i.e., for every query vector $q' \in [0, 1]^d$, p_5 scores lower than p_2. Hence, any non-result record would have to overtake p_5 before it can reach p_2 in score. That is, we can safely disregard all result records that dominate at least another record in R. This strategy prunes p_2, p_1, p_4 (the first dominates p_5 and the other two p_6).

In summary, we disregard result records that (i) lie inside the convex hull of R or (ii) dominate at least one other record in R. We denote by R^- the remaining result records. In our example, $R^- = \{p_5, p_6\}$. Subsequent processing follows SP, CP or FP.

SP and CP produce the GIR$_i$ region for each $p_i \in R^-$ in the same way as in Section 5, and report $\bigcap_{i=1}^{i=|R^-|}$ GIR$_i$ as GIR*. Note that $\mathcal{SL}$ and $\mathcal{SL} \cap \mathcal{CH}$ over the non-result records (in SP and CP, respectively) are computed once and used for all GIR$_i$ derivations.

FP, in its first step, considers the set T of non-result records encountered during top-k computation. For every $p_i \in R^-$, it computes the set $\mathcal{F}_i$ of facets that are incident to p_i (on the convex hull of set $\{p_i\} \cup T$). This is done in the same fashion as in Sections 6.2 or 6.3.1, depending on dimensionality.

In the second step, FP maintains the $\mathcal{F}_i$ sets concurrently as it fetches new non-result records from the disk. The process is similar to Section 6. The main difference is that index nodes (that are popped from the search heap) are only pruned if they lie below all the facets in every $\mathcal{F}_i$ set. Also, each record p fetched from disk is checked against all the $\mathcal{F}_i$ sets and used to update those that include at least one facet below p. When the search heap becomes empty, each $\mathcal{F}_i$ set is used to derive the respective GIR$_i$ region. Finally, FP reports GIR$^* = \bigcap_{i=1}^{i=|R^-|}$ GIR$_i$.

Continuing our example, Figure 12(b) illustrates facet sets $\mathcal{F}_5$ and $\mathcal{F}_6$. Each facet in $\mathcal{F}_5$ determines a half-space of the form $(p_5 - p) \cdot q' \geq 0$. The intersection of these half-spaces is GIR$_5$. Region GIR$_6$ is derived similarly, and FP reports GIR* = GIR$_5 \cap$ GIR$_6$.

7.2 Non-Linear Scoring Functions

Our main focus in this paper is on linear scoring functions. While CP and FP (rely on convex hull properties that) may not extend to more general function types, SP can handle a broader class of functions. The following discussion considers the original (order-sensitive) GIR, but it translates easily to the GIR* context too.

There are two components in SP, namely, (i) pruning non-result records, and (ii) using the remaining non-result records to form the GIR. SP pruning applies to any monotone[8] scoring function. In other words, the only non-result records that could overtake the k-th record in R, under any such function, are guaranteed to belong to the skyline of $D \backslash R$ [21].

Identifying the non-result records that could affect the GIR helps to limit the number of conditions in Definition 1. Forming the GIR, however, requires translating these conditions to a locus in query space. If the scoring function is of the form $\mathcal{S}(p, q) = \sum_{i=1}^{d} w_i g_i(p)$, where each $g_i(p)$ is a function over the attributes of p, the GIR is derived using half-space intersection as per normal. To see this, the GIR is defined by conditions of the form $\mathcal{S}(p, q') \geq \mathcal{S}(p', q')$ for various pairs of records p, p'. Condition $\mathcal{S}(p, q') \geq \mathcal{S}(p', q')$ can be rewritten as $\sum_{i=1}^{d} w_i g_i(p) \geq \sum_{i=1}^{d} w_i g_i(p') \Rightarrow \sum_{i=1}^{d} w_i(g_i(p) - g_i(p')) \geq 0$. Since we are comparing specific records p and p', the factors $(g_i(p) - g_i(p'))$ are constants, and thus the condition corresponds to a half-space in query space. That is, Phase 1 and Phase 2 may employ plain half-space intersection to produce the GIR.

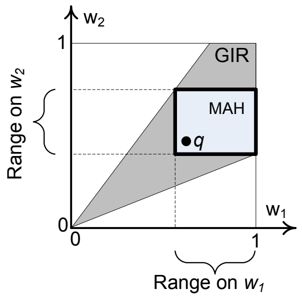
For scoring functions that do not belong to the above category, conditions of the form $\mathcal{S}(p, q') \geq \mathcal{S}(p', q')$ no longer correspond to half-spaces. This implies that the GIR is no longer a convex polytope but a general convex set. Exact representation of the GIR in such cases is computationally expensive or not possible at all, which would call for approximate GIR representation techniques, such as polytope approximation, Monte Carlo simulation, etc [30].

7.3 GIR Visualization

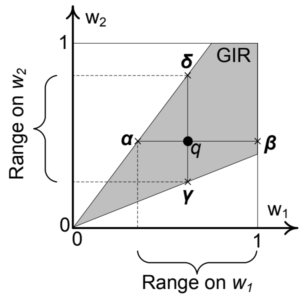
One of the GIR applications in Introduction is to give the user a sense of the weight shift required to induce a change in the top-k result. Being a d-dimensional polytope, GIR is challenging to visualize for $d > 2$. We describe two possible visualization options.

Assuming that the GIR is already derived using any of our methods, the first visualization technique computes the maximum-volume axis-parallel hyper-rectangle (MAH) that (i) contains the

[8]We follow the convention that the larger a record's attributes the higher its score. A scoring function $\mathcal{S}(p, q)$ is monotone iff for any dimension $i \in [1, d]$ and for any pair of records p, p' with $p.x_i \geq p'.x_i$ and $p.x_j = p'.x_j \ \forall j \neq i$, it holds that $\mathcal{S}(p, q) \geq \mathcal{S}(p', q)$.



(a) MAH



(b) Interactive projection

Figure 13: GIR visualization ($d = 2$, query space)

query vector q and (ii) lies completely inside the GIR. This is an instance of the *bichromatic rectangle problem*, for which several algorithms are available [2, 16]. Figure 13(a) shows the MAH in a 2-dimensional query space. The MAH can be visualized easily by projecting its sides on the different axes, producing for example bounds like those in Figure 1. The advantage of this approach is that the bounds are fixed as long as the query vector remains inside the MAH. The disadvantage is that the MAH is a subset of the GIR.

An alternative is the interactive projection approach, which does not sacrifice maximality (i.e., it allows exploration of the full extent of the GIR) but requires on-the-fly readjustment of the bounds plotted on the interface. Consider the query vector in Figure 13(b). We first find the horizontal projections of q on the GIR, i.e., points α and β, and map them on the w_1 axis to derive an upper and lower bound for w_1 like those in Figure 1. Similarly, we project q vertically on the GIR (getting points γ and δ) and produce the bounds for w_2. Note that the derived ranges are equivalent to the LIRs in [24]. Should the user shift the query vector (by varying one or multiple weights), we may interactively re-project the new location of q on the GIR, and redraw on-the-fly the new permissible ranges for each factor. That is, as the user shifts q (within the GIR), she sees the bounds for each factor being adjusted in real time.

8. EXPERIMENTS

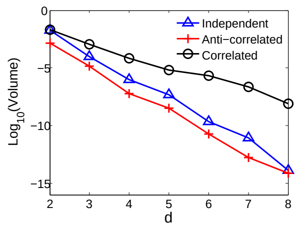
In this section we evaluate the efficiency of the SP, CP, and FP algorithms, using synthetic and real datasets. The synthetic datasets include *Independent* (IND), *Correlated* (COR), and *Anti-correlated* (ANTI), which are standard benchmarks for preference-based queries [8]. IND is uniformly and independently distributed. In COR, records that have a large value in one dimension tend to have large values in the other dimensions too. In ANTI, a record with a large value in one dimension tends to have small values in the rest. We also use real datasets HOUSE and HOTEL (from *ipums.org* and *hotelsbase.org*, respectively). HOUSE contains 315,265 records, each with six attributes representing an American family's expenditure in gas, electricity, water, heating, insurance, and property tax. HOTEL contains 418,843 hotel records with four attributes, namely stars, price, number of rooms, and number of facilities. All attributes are normalized to $[0, 1]$. The datasets are indexed by an R*-tree using 4KByte disk pages.

In the default setting, we place data and indices on the disk and evaluate performance in terms of CPU and I/O time[9]. However, the CPU charts in isolation indirectly also evaluate the scenario where data and indices are kept in memory. Unless otherwise specified, we compute the order-sensitive GIR. We present total CPU and I/O times, accounting for Phases 1 and 2. A buffer for disk pages cannot improve I/O time, since none of the methods fetches the same index or data page twice. Thus, we do not use one. All methods are implemented in C++ and use the Qhull library (*qhull.org*) for half-space intersection. Experiments are run on a PC with Intel Core2Duo 3GHz CPU. Table 2 summarizes the investigated parameters, along with their tested values and defaults (in bold). With the real datasets we control only the last parameter (i.e., k). Each reported measurement is the average over 100 random queries.

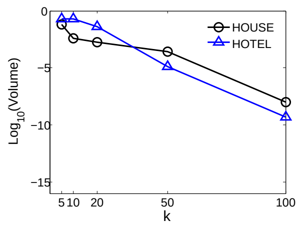
[9]Note that SP and CP have identical I/O cost, because they access the disk through the same BBS process.

Parameter	Range of values
Dimensionality, d	2, 3, **4**, 5, 6, 7, 8
Dataset cardinality, n	0.5M, **1M**, 5M, 10M, 20M
Top-k result size, k	5, 10, **20**, 50, 100

Table 2: Experiment Parameters



(a) Varying d (Synthetic)



(b) Varying k (Real data)

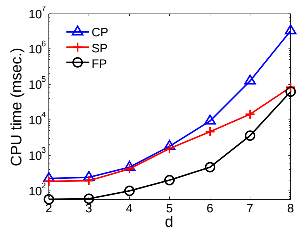
Figure 14: Ratio of GIR volume to query space volume

We first provide insight into the nature of the GIR. In Figure 14 we present the ratio of GIR volume to the volume of the query space, which coincides with the sensitivity measure discussed in the Introduction and the LIK probability in [30]. In Figure 14(a) we use synthetic data and vary d. The GIR volume drops exponentially with d, and Figure 8(b) provides the reason. As d increases, so does the number of facets incident to p_k, which implies that more conditions (half-spaces) bound the GIR. The GIR is the largest in COR and the smallest in ANTI, because COR has the fewest incident facets among our synthetic data, while ANTI has the most (as shown in Figure 8(b)). The trends in Figure 14(a) also reveal an interesting fact about the nature of the top-k query itself; the alternative top-k results become dramatically less distinguishable as d grows. In Figure 14(b) we plot the volume ratio versus k for the real data. A larger k implies more half-spaces induced from the first set of conditions in Definition 1, leading to a smaller GIR.

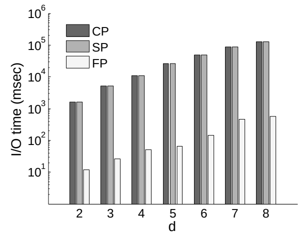
In Figure 15 we study the effect of dimensionality d on the performance of SP, CP, and FP, using synthetic data. All the charts for this experiment are in logarithmic scale. FP outperforms SP and CP in all cases, with SP being the runner-up. The largest differences are observed for ANTI, where FP takes 53 to 2700 times shorter I/O time than SP, and 1.3 to 47 times shorter CPU time. The difference is smaller in COR (because there are fewer skyline records than in IND and ANTI), with FP, however, still performing 9.6 to 224 times fewer I/Os and 1.8 to 24 times fewer computations. Interestingly, the CPU time of CP is longer than SP. Although CP prunes more records, its expensive convex hull computation outweighs the benefits of pruning (an issue discussed at the end of Section 5.3).

In Figure 16 we use IND and investigate the effect of dataset cardinality n, varying it from 0.5M to 20M tuples. CPU and I/O times naturally increase with n in all methods. The important finding is that FP scales much better with cardinality, as a result of focusing only on the (relatively few) convex hull facets that are incident to p_k. In terms of I/O cost, it outperforms the runner-up (SP) by 460 to 1748 times, and by 2.8 to 16.5 times in terms of CPU cost.

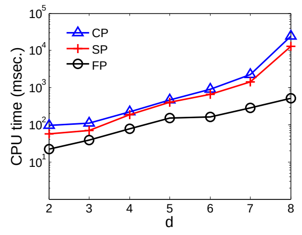
In Figure 17 we assess the effect of k using the real datasets. A larger k implies more records in T, i.e., more non-result records encountered during top-k computation by BRS. The larger T leads



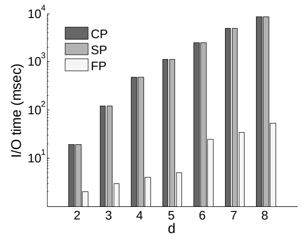
(a) CPU time (IND)



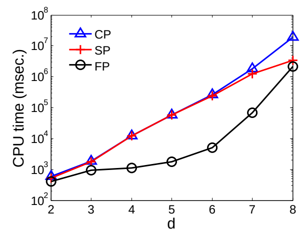
(b) I/O time (IND)



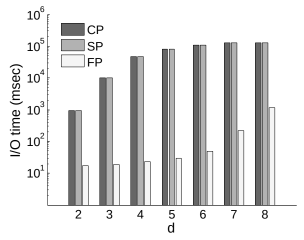
(c) CPU time (COR)



(d) I/O time (COR)

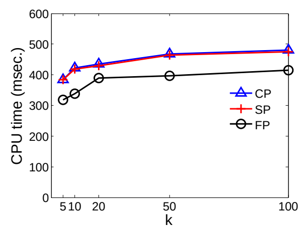


(e) CPU time (ANTI)

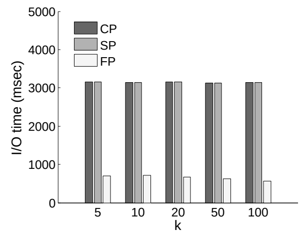


(f) I/O time (ANTI)

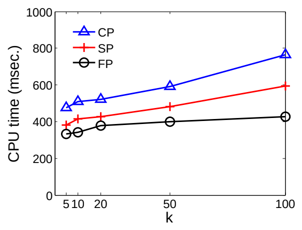
Figure 15: Effect of dimensionality d for synthetic data



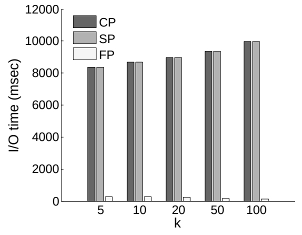
(a) CPU time (HOTEL)



(b) I/O time (HOTEL)

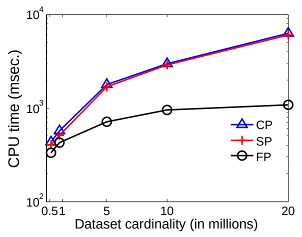


(c) CPU time (HOUSE)

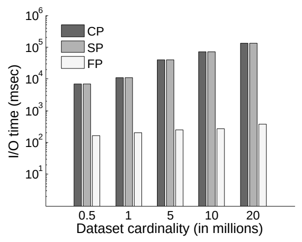


(d) I/O time (HOUSE)

Figure 17: Effect of k for real data



(a) CPU time



(b) I/O time

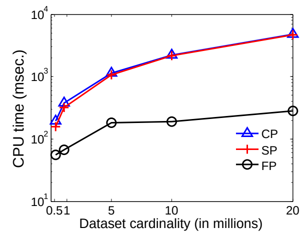
Figure 18: Order-insensitive GIR, effect of n (IND)

to an increase in CPU time. On the other hand, the effect of k on I/O cost involves two conflicting factors. A larger T implies that most critical records (in FP) and skyline records (in SP/CP) have already been fetched from disk (by BRS). This leads to a slight decrease in I/O cost for all methods in HOTEL. In HOUSE, however, due to its higher dimensionality (six instead of four) and different distribution, the inclusion of more records in the top-k result (and thus their exclusion from $D\backslash R$) deprives the skyline computation module (BBS) of records with high dominating/pruning power and "widens" the skyline, thus raising the I/O cost for SP and CP. In contrast, FP is independent of the skyline, and its I/O cost slightly decreases with k in this dataset as well.

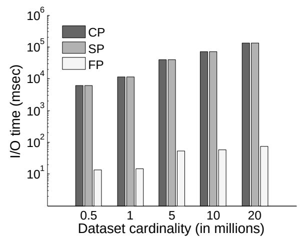
In Figure 18 we evaluate our algorithms for the computation of *order-insensitive* GIR. We set all parameters to their defaults and vary n (for IND data). The trends are similar to Figure 16; however, the cost of all methods increases. The reason is that, as explained in Section 7.1, multiple result records need to be considered against the non-results (as opposed to just considering p_k against them).

Returning to the default, order-sensitive GIR, in Figure 19 we consider *non-linear* scoring functions. Using HOTEL and varying k, we investigate the performance of SP for (monotone) functions $\mathcal{S}(p,q) = w_1x_1^4 + w_2x_2^3 + w_3x_3^2 + w_4x_4^1$ and $\mathcal{S}(p,q) = w_1x_1^2 + w_2e^{x_2} + w_3\log x_3 + w_4\sqrt{x_4}$ (recall that HOTEL is 4-dimensional). We label these functions as "Polynomial" and "Mixed", respectively. For comparison, we also include "Linear".

SP performance is similar for all functions. That is because skyline computation by BBS is independent of the function type (thus the comparable I/O cost), which in turn leads to a similar number of half-spaces to intersect for GIR derivation (thus the comparable CPU time). Results with other monotone functions are similar and omitted in order not to clutter the charts.

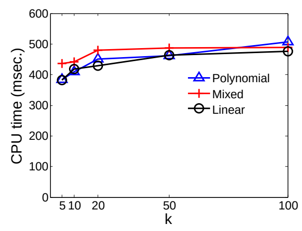


(a) CPU time

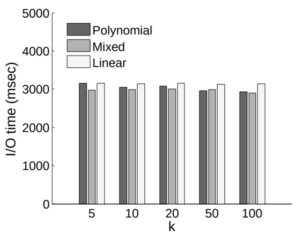


(b) I/O time

Figure 16: Effect of dataset cardinality n (IND)



(a) CPU time



(b) I/O time

Figure 19: Non-linear scoring functions, effect of k (HOTEL)

9. CONCLUSIONS

In this paper we study the problem of global immutable region (GIR) computation. Assuming a top-k query with a linear scoring function, the GIR indicates all the possible weight settings that produce exactly the same result as the original query. The GIR can be used as a guide for query weight refinement, as a sensitivity measure, and as a means for result caching. We propose a suite of scalable algorithms that exploit the geometric properties of the problem to achieve efficient GIR computation. A direction for future work is to extend the GIR notion to other query types and complex scoring functions.

10. ACKNOWLEDGMENTS

This work was supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office. Jilian Zhang was also supported by travel grant 61363009 from China NSF.

11. REFERENCES

[1] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD Conference*, pages 28–39, 2003.

[2] J. Backer and J. M. Keil. The bichromatic rectangle problem in high dimensions. In *CCCG*, pages 157–160, 2009.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[4] 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.

[5] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *JACM*, 25(4):536–543, 1978.

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

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

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

[9] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry*, 16(4):361–368, 1996.

[10] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *JACM*, 17(1):78–86, 1970.

[11] 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*, pages 391–402, 2000.

[12] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *FOCS*, pages 29–38, 1991.

[13] H. Christopher Frey and S. R. Patil. Identification and Review of Sensitivity Analysis Methods. *Risk Analysis*, 22(3):553–578, 2002.

[14] K. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185–212, 1993.

[15] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *VLDB*, pages 451–462, 2006.

[16] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[17] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

[18] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

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

[20] D. M. Hamby. A review of techniques for parameter sensitivity analysis of environment models. *Environmental Monitoring and Assessment*, 32(2):135–154, 1994.

[21] 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.

[22] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. In *PVLDB*, pages 301–312, 2008.

[23] J. Matousek and O. Schwarzkopf. Linear optimization queries. In *ACM Symposium on Computational Geometry*, pages 16–25, 1992.

[24] K. Mouratidis and H. Pang. Computing immutable regions for subspace top-k queries. In *PVLDB*, pages 73–84, 2013.

[25] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The v*-diagram: a query-dependent approach to moving knn queries. In *PVLDB*, pages 1095–1106, 2008.

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

[27] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers*, 51(10):1124–1140, 2002.

[28] A. Saltelli, K. Chan, E. M. Scott, et al. *Sensitivity analysis*. Wiley New York, 2000.

[29] A. Saltelli, S. Tarantola, and K.-S. Chan. A quantitative model-independent method for global sensitivity analysis of model output. *Technometrics*, 41(1):39–56, 1999.

[30] 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.

[31] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[32] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3):424–445, 2007.

[33] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, pages 277–288, 2003.

[34] 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.

[35] A. Vlachou, C. Doulkeridis, K. Norvag, and Y. Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *SIGMOD*, pages 481–492, 2013.

[36] M. Xie, L. V. S. Lakshmanan, and P. T. Wood. Efficient top-k query answering using cached views. In *EDBT*, pages 489–500, 2013.