Representative Subsets for Preference Queries

by

Sean Chester
B.Sc., University of Victoria, 2007
M.Sc., University of Victoria, 2009

A Dissertation Submitted in Partial Fulfilment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Representative Subsets for Preference Queries

by

Sean Chester
B.Sc., University of Victoria, 2007
M.Sc., University of Victoria, 2009

Supervisory Committee

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Co-supervisor
(Department of Computer Science)

Dr. Sue Whitesides, Co-supervisor
(Department of Computer Science)

Dr. Hong-Chuan Yang, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Co-supervisor
(Department of Computer Science)

Dr. Sue Whitesides, Co-supervisor
(Department of Computer Science)

Dr. Hong-Chuan Yang, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

We focus on the two overlapping areas of preference queries and dataset summarization. A (linear) preference query specifies the relative importance of the attributes in a dataset and asks for the tuples that best match those preferences. Dataset summarization is the task of representing an entire dataset by a small, representative subset. Within these areas, we focus on three important sub-problems, significantly advancing the state-of-the-art in each.

We begin with an investigation into a new formulation of preference queries, identifying a neglected and important subclass that we call *threshold projection queries*. While literature typically constrains the attribute preferences (which are real-valued weights) such that their sum is one, we show that this introduces bias when querying by threshold rather than cardinality. Using projection, rather than inner product as in that literature, removes the bias. We then give algorithms for building and querying indices for this class of query, based, in the general case, on geometric duality and halfspace range searching, and, in an important special case, on stereographic projection.

In the second part of the dissertation, we investigate the *monochromatic reverse top-k* (mRTOP) query in two dimensions. A mRTOP query asks for, given a tuple and a dataset, the linear preference queries on the dataset that will include the given tuple. Towards this goal, we consider the novel scenario of building an index to support mRTOP queries, using geometric duality and plane sweep. We show theoretically and empirically that the index is quick to build, small on disk, and very efficient at answering mRTOP queries. As a corollary to these efforts, we defined the *top-k rank contour*, which encodes the k-ranked tuple for every possible linear preference query. This is tremendously useful in answering mRTOP queries, but also, we posit, of significant independent interest for its relation to myriad related linear preference query problems. Intuitively, the top-k rank contour is the minimum possible representation of knowledge needed to identify the k-ranked tuple for any query, without *apriori* knowledge of that query.

We also introduce *k-regret minimizing sets*, a very succinct approximation of a numeric dataset. The purpose of the approximation is to represent the entire dataset by just a small subset that nonetheless will contain a tuple within or near to the top-k for any linear preference query. We show that the problem of finding k-regret minimizing sets—and, indeed, the problem in literature that it generalizes—is NP-Hard. Still, for the special case of two dimensions, we provide a fast, exact algorithm based on the *top-k rank contour*. For arbitrary dimension, we introduce a novel greedy algorithm based on linear programming and randomization that does excellently in our empirical investigation.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to sincerely thank:

my supervisor of six years, Dr. Alex Thomo, for being such a superb mentor, creating me as a researcher, from first encouraging and inspiring me to pursue graduate studies, to discovering how to provide the supportive environment, specific to me, that would best stimulate learning and growth, to instilling in me a passion and excitement for research and teaching me how to formulate and revise my ideas and work;

my co-supervisor, Dr. Venkatesh Srinivasan, for the equally superb mentorship, helping me to round out as a researcher by developing my theoretical toolkit with long meetings spent verifying proofs and longer meetings spent clarifying prose, and for the guidance and re-assurance to combat my inevitable periods of uncertainty and self-doubt;

my beautiful wife, Andrea, for the tremendous love and support, being my organizational safety net that consistently rescues me from my own absent-mindedness – a forgotten computer cable here and an overwhelmingly Mandarin-only hotel stay there;

my dear friend, Simon Pearson, for the extensive friendship and personal growth – transforming me from a monkey slapping together thousand line main() programmes to an engineer of tightly coupled code and tests, being my personal consultant always more aware than I of my path in any endeavour, thorough and enthusiastic in fuelling my humility, and ever ready to connect on a whim for a coffee, a walk, a run, a movie, or whichever environs can best lend levity to life's and work's challenges;

my co-authors, particularly Dr. Gautam Srivastava, for the enriching experience of collaboration, with which I developed an exciting and impactful concurrent research interest, learned the importance of a social aspect to research, and became a more qualified candidate, in terms of both publications and research skills – it is perhaps no coincidence that a year after helping Gautam with the "Complexity of Social Network Anonymization" that I proved hardness in my research, as well;

my gracious brother, Steven, for the patience and accomodation to, at a moment's notice, repeatedly, and out of the blue, lend me computing power when my little netbook suddenly would not suffice, and for the standing invitation to scream at hockey on the telly with him rather than at my bug-ridden, in-development code by myself;

my parents, Mark and Marian, for teaching me with groups of raisins on the kitchen table the *concepts* of subtraction and multiplication and with books at bedtime my basic literacy, the foundational skills that set up twenty-five years later the most elaborate of proofs in this dissertation;

my many close running friends, for the balance, perspective, camaraderie, and transferable life lessons best acquired through sweat;

the administrative staff in the department, Wendy Beggs and Nancy Chan, for their attentiveness and care, and without whom I would never have cleared enough red tape to complete my degree "on-time" and collect reimbursement for conference travel;

my recent contractees, Mark Nelson and Dr. Eleanor Setton, for the diversifying experiences that come with contract work, providing me not just with the extra finances to top up my support, but also an engagement with other disciplines and with industry to better understand the practical needs outside academia;

my examining committee, my co-supervisor Dr. Whitesides, and Drs. Yang and Lall, for the helpful comments that, while focused on improving the quality of this manuscript, identify trends of weakness in my writing skills in general, which in turn will help me to improve as a researcher; and, finally,

the varied, creative, artisinal coffeehouses of Victoria, whose napkins were my "proving grounds" as a young doctoral researcher.

DEDICATION

To the wonderful world of Science.

Chapter 1

Introduction

Modern datasets are of unprecedented size. The reasons for this are diverse, ranging from the ease of collecting sensor data to the explosion of consumer choice to the modern wealth of user-contributed content. But despite all this data, there is not much one can do if overwhelmed by it all. Without some computational intervention, the analysis task is infeasible. But on unseen data, even defining the computational task can be challenging.

So, an alternative is to summarize the entire dataset with a much smaller subset that well represents the broad spectrum in the data. This can serve a number of purposes. For one, summarization can be a first step in finding the most interesting tuples in a dataset. If that representative subset is sufficiently small, it can even be presented to the user as a 'guide' to what the best tuples in the dataset look like, and can help the user in formulating his query preferences if they are previously unknown. Perhaps more interestingly, performing a query on a subset of the data can be much faster and, depending on the means by which the summarization was performed, can still produce the same answer, leading to asymptotic improvement in query performance.

Consider an archetypal example, navigating through an entire dataset of available hotels, trying to decide on the best option. While selection based on location helps, it does not narrow down the options by so much. For example, at the time of writing there are 363 hotels within one mile of Madison Square Garden in New York, according to `http://yelp.com`. Showing a few distinctive hotels that represent the broad spectrum of what is available can help to narrow down the choices.

This example can also be used to illustrate another interesting approach to helping a user discover the tuples of relevance in a large dataset. While James Bond will expect a hotel that is expensive, flashy and right beside the venue of interest, I would content myself with whatever happens to be the cheapest, even if it is five miles away; different users have

different personal interests and so it does not make much sense to present both with the same query results if you know what those interests are. This is the idea behind the well established and popular research area of *preference queries*: a user is modelled by a query that captures his relative interests so that the results he is presented with are *personalized*. It is a broad area, and within it we focus specifically on cases where the user is modelled by a linear function that can be applied to the database attributes. How that modelling is done is outside the scope of this dissertation—we are interested in the computational challenges in servicing this model.

The interplay between these two research areas, dataset summarization and preference querying, is especially fascinating. When the query model is known, the basic assumptions can be richer, and so the computational tasks can be more sophisticated. In the case of summarizing datasets for preference queries in particular, one can directly measure the efficacy of a particular representative subset and thus compare subsets against each other. Furthermore, those representative subsets can help to more efficiently resolve the preference queries. These are the high-level goals in this dissertation, to better support efficient resolution of preference queries using subsets of different types.

We pursue this goal in three directions that we outline in the remainder of this chapter: formulating threshold projection queries; answering monochromatic reverse top-k queries; and discovering k-regret minimizing sets. To help the reader, we have made the main chapters self-contained, each with their own sections of relevant preliminaries, technical contributions, engagement with literature, and conclusions, in that order. Formal definitions and result statements are deferred until needed. The reader is, of course, welcome at any point to jump from a particular subsection of the introduction to the relevant chapter if he is particularly inclined. That said, it is worth reviewing Section 2.1 first, as it introduces some of the basic algebraic and geometric concepts that are common throughout the entire dissertation.

1.1 Threshold Projection Queries

Consider how you would express a query to identify employees whose salaries are especially large relative to their age. For such a purpose, many applications feature queries wherein attributes are combined into a single score using a linear function, such as $\mathrm{score} = a * \mathrm{salary} + b * \mathrm{age}$. The result set is then determined based on that score. Queries 1 and 2 below are examples of this. To emphasise one attribute over another in the query requires simply adjusting the coefficients in the linear function (i.e., a and b). The expression of a

query in terms of these coefficients is often called a *preference query* when the coefficients can be thought of as representing a user's preferences with respect to each attribute.

Query 1 (Young, high-paid employees)**.**
*SELECT **
FROM Employees
ORDER BY $3 * \text{salary} - 4 * \text{age}$ *DESC*
LIMIT 10;

Query 1 is an example of a typical top-k query, which, for more precision, we call a top-k *linear optimisation query* (LOQ). It selects the 10 employees from a table of employees who have the highest weighted age and salary. This section explains why in this *preference-based* setting Query 2 is often better than Query 1. Moreover, we describe how to index numeric datasets to support queries of the form in Query 2. We call these *threshold projection queries* (TPQs), because, in a "tuples-as-vectors" perspective, the result of the query is those vectors with a *projection* onto the query greater than a pre-specified threshold.

Query 2 (Young, high-paid employees)**.**
*SELECT **
FROM Employees
WHERE $\text{proj}(\text{salary}, \text{age}, \langle 3, -4\rangle) > \tau$*;*

We adopt the "tuples-as-vectors" perspective throughout this work, in which a tuple $(a_1, \ldots, a_d)$ is alternately represented as a vector $\vec{\text{v}} = \langle a_1, \ldots, a_d\rangle$. By relaxedly referring to a tuple as a vector (or, also, a point), and switching into the appropriate mathematical context, the discussion is greatly simplified. So, under this perspective, a LOQ ranks tuples by the size of their dot product with a query vector $\vec{\text{q}}$ from the same domain. (In the case of Query 1, $\vec{\text{q}} = \langle 3, -4\rangle$.) Typically, the result of a LOQ is limited to the k highest ranked tuples (a top-k query).

In Chapter 2, we propose a new approach to preference queries based on vector projection rather than dot product in order to address short-comings of LOQs. Also, we justify the use for some contexts of *threshold queries*, in which the result set is determined by minimum scores, rather than top-k, in which the result set is capped by a predetermined cardinality.

1.1.1 Dot Product vs. Projection

First, consider that the output is determined by thresholds rather than cardinality limits (we justify this shortly). Then, the result $\mathcal{R}$ of a threshold LOQ on a dataset $\mathcal{D}$ of vectors, given as input a query vector $\vec{\mathrm{q}}$ and a threshold τ is:

$$\mathcal{R} = \{\vec{\mathrm{v}} \in \mathcal{D} : \vec{\mathrm{v}} \cdot \vec{\mathrm{q}} \geq \tau\}.$$

This problem formulation suggests some difficulties that arise out of using dot product as a measure of similarity. First, there are two user-defined inputs, the modification of which have inversely related effects: one can adjust either τ or $||\vec{\mathrm{q}}||$ to alter the result set cardinality without altering the intent. These effects cancel each other because τ and $||\vec{\mathrm{q}}||$ are inversely related in the query condition. This makes it very difficult to prune the dataset based on one factor, because, for example, no matter how small one chooses τ, there are choices for $||\vec{\mathrm{q}}||$ that will lead to vectors in $\mathcal{D}$ being part of the result set $\mathcal{R}$.

This concern is recognised in the LOQ literature: there, first, an assumption is introduced that only the positive quadrant is of interest and, second, $\vec{\mathrm{q}}$ is constrained such that $\sum q_i = 1$. We call this *constrained LOQ*. This fixes the domain of possible queries to be part of a (d-1)-hyperplane rather than $\mathbb{R}^d$. In the case of two dimensions, this means that any possible query lies on the line segment $y = 1 - x, x \in [0, 1]$. This suffices when the result set cardinality has been fixed in advance (top-k).

However, this solution introduces a new problem for threshold queries: it creates a bias towards queries farther (angularly speaking) from the axes (See Figure 1.1). Because a larger magnitude of the query inversely affects $|\mathcal{R}|$ in a LOQ, queries angularly nearer to the axes have the same effect as a decreased τ since they have larger magnitudes (near to one). On the other hand, for a query angularly equidistant from the axes, $\vec{\mathrm{q}} = \langle \frac{1}{d}, \ldots, \frac{1}{d} \rangle$, and $||\vec{\mathrm{q}}|| = \sqrt{d(1/d^2)} = d^{-\frac{1}{2}}$. That is to say, in d dimensions queries are $\sqrt{d}$-fold less selective near the axes than maximally distant from the axes. Hence, to use the system effectively, one has to understand within the context of his domain the interplay between query direction and bias and how to appropriately adapt the threshold to reflect the revised objectivity of any new query options. The result sets of different query vectors with the same threshold are incomparable.

Hence we propose the use of vector projection (TPQ) instead of dot product. The im-

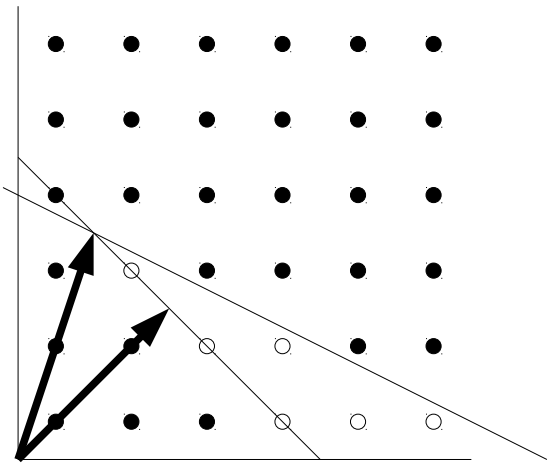


Figure 1.1: *The bias introduced by constrained LOQ*. Of the uniformly distributed points, the white points will be returned by the non-partisan query $(\langle .5, .5\rangle)$ but not the query nearer the y-axis ($\langle .25, .75\rangle$). The non-partisan query has a larger result set simply because the query vector has smaller magnitude.

partiality towards $||\vec{\mathrm{q}}||$ is built into the definition of projection, since the result set becomes:

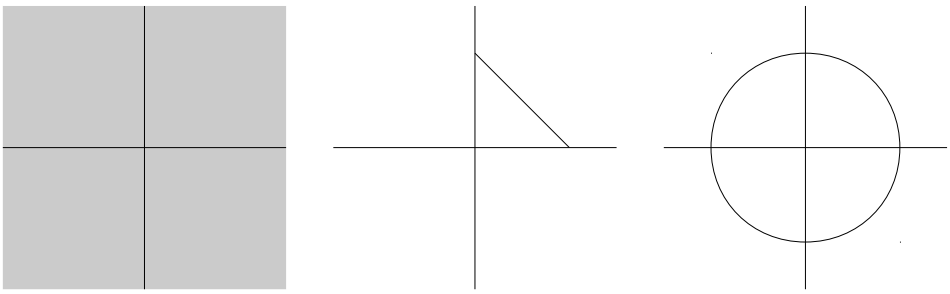
$$\mathcal{R} = \{\vec{\mathrm{v}} \in \mathcal{D} : \frac{\vec{\mathrm{v}} \cdot \vec{\mathrm{q}}}{||\vec{\mathrm{q}}||} \geq \tau\}.$$

This new definition can be interpreted alternatively as modifying the constraints placed on the set of possible queries, instead permitting only unit vectors. Figure 1.2 illustrates the possible queries in two dimensions for LOQ, constrained LOQ, and TPQ. For LOQ, possible queries span the entirety of $\mathbb{R}^d$. By constraining the queries, that space is reduced to the line segment:

$$x_d = 1 - \sum_{1}^{d-1} x_i, x_i \geq 0.$$

For TPQ, on the other hand, the queries span:

$$\mathcal{U} = \{\vec{\mathrm{q}} \in \mathbb{R}^d : ||\vec{\mathrm{q}}|| = 1\},$$



(a) LOQ (b) Constrained LOQ (c) TPQ

Figure 1.2: *Query spaces for various classes of preference queries*

which forms the (d-1)-sphere[1]:

$$x_d^2 = 1 - \sum_{1}^{d-1} x_i^2.$$

1.1.2 Thresholds vs. Top-k

Studying the threshold variant is very interesting. There is a misconception that top-k and threshold are equivalent for suitably chosen values of k and/or τ, but this is not true. In fact, the threshold variant is more flexible. Consider trying to transform the following threshold query into a top-k query:

Query 3 (Suspicious recent transactions in a region)**.**
*SELECT **
FROM Transactions NATURAL JOIN Store_Locations
WHERE $\mathrm{proj}(\mathrm{latitude}, \mathrm{longitude}, \langle 3, 4\rangle) > \tau_1$
AND $\mathrm{proj}(\mathrm{trans\_value}, \mathrm{trans\_frequency}, \langle 1, 1\rangle) > \tau_2$
AND $\mathrm{proj}(\mathrm{timestamp}, \langle 1\rangle) > \tau_3$*;*

As a threshold problem this is very realistic for the DBMS to handle: it executes the three conditions as separate queries using appropriately built indices and then performs an intersection with the resultant tuple pointers to execute the conjunction. Likewise with

[1]Throughout this paper, we refer to the sphere in d dimensions as the (d-1)-sphere. For example, the outer surface of the Earth is (approximately) a 2-sphere.

pointer union for disjunction. Here, the assumption that one can simply replace the threshold with an appropriately chosen value of k to obtain the same result set is incorrect: there are numerous combinations of τ_1 and τ_2 that will both yield $|\mathcal{R}| = k$, but they will not necessarily produce the same result sets. The threshold variants are more flexible than the top-k variants in this sense and by using pointer intersection/union, the indices need not be defined any differently in order to support conjoined and disjoined conditions.

Additionally, it often makes sense to issue thresholds rather than result set cardinalities because an appropriate cardinality is not always clear. Query 3, for example, could be issued by a credit card agency seeking to determine fraudulent transactions based on a linear function of transaction frequency and value. The number of suspicious transactions that occurred in the given location within a specified time period is variable and not known to the analyst in advance of issuing the query, so there is not a clear argument for having the analyst guess in advance how many suspicious transactions will have taken place. On the other hand, establishing a suitable threshold is reasonable, because the analyst can utilise his domain expertise and prior experience.

1.1.3 Rendering TPQ Feasible

Certainly, it is undesirable to scan an entire relation for every query issued, if avoidable. So, we would like to design an effective *index*, a choice of physical dataset layout that improves the efficiency with which we can respond to the queries. But to derive a suitable index for TPQ queries is non-trivial. Indexing vectors in their natural form is not useful without *apriori* knowledge of the query vector. The difficulty is that an index organises tuples so that similar tuples are near each other (logically in the case of secondary indices and physically in the case of primary indices), but the similarity of vectors to one another does not necessarily imply the similarity of their projections onto arbitrary query vectors. Nor does the similarity of their projections onto one query imply the similarity of their projections onto other, arbitrary query vectors.

As an example, consider two vectors $\vec{\mathrm{u}}$ and $\vec{\mathrm{v}}$. The projections of $\vec{\mathrm{u}}$ and of $\vec{\mathrm{v}}$ onto $\vec{\mathrm{q}} = \vec{\mathrm{u}} + \vec{\mathrm{v}}$ are quite similar (or at least both of positive sign). However, the projections of the same vectors onto a query $\vec{\mathrm{q}}'$ orthogonal to $\vec{\mathrm{q}}$ are very dissimilar (of opposite signs). This example demonstrates that efforts to pre-organise the vectors can be substantially thwarted depending on user query choices.

A major contribution that we present is in addressing this challenge by using geometric transformations of the query problem. In particular, we introduce the notion of a *cap*

(Section 2.2), the geometric representation of all unit queries for which the vector should be returned. By employing this insight, the problem becomes to index caps to efficiently resolve the equivalent problem, *in which caps does one find the query?*

For the general case (Section 2.3), wherein the dimension is arbitrary and there are no constraints on the threshold, we demonstrate how the application of a duality transform can permit responding to this equivalent query by solving a *halfspace range searching* problem. The advantage of this technique is that halfspace range searching has been optimally solved for external memory. Consequently, we asymptotically improve upon the sequential scan alternative.

We then consider the specialised scenario of static thresholds in two and three dimensions (Section 2.4), which we consider an interesting subcase because, for example, user interfaces may often constrain a user's selections to a predefined finite set. We demonstrate that under these conditions, performance can be markedly improved. We employ stereographic projection on the caps and index their images in a spatial index, thus improving asymptotic query performance from sub-linear in the general case to logarithmic in this special case.

1.2 Monochromatic Reverse Top-k Queries and Top-k Rank Contours

In this age of arbitrarily large datasets, personalizing query results for users with LOQs has become ubiquitous. Consider again the common approach of querying a dataset $\mathcal{D}$ of n numeric tuples $(a_1 \in \mathbb{R}, \ldots, a_d \in \mathbb{R})$. The top-k query models the user with an ordered list of weights $(w_0, \ldots, w_{d-1})$, representing his degree of "personal preference" for each of the d attributes of $\mathcal{D}$. In executing the query, each tuple $t \in \mathcal{D}$ is assigned a score, $\mathrm{score}(t) = w_0 a_0 + \ldots + w_{d-1} a_{d-1}$, and the k tuples with highest score are presented as the user's personalized query results.

In Chapter 3, we consider these top-k queries from the perspective of the tuple rather than the user. A tuple $t \in \mathcal{D}$ is only relevant if it is the response to some top-k query. Its relevance is proportional to the breadth of queries for which it is returned. A *monochromatic reverse* top-k (mRTOP) query [46] computes that breadth. Given a (possibly new) tuple of interest, q, a reverse top-k query reports the set of LOQs on $\mathcal{D} \cup \{q\}$ for which q is in the result set.

Table 1.1 gives a small example of two traditional top-k queries in the rightmost columns,

			query a: $(0.75, 0.25)$		query b: $(0.25, 0.75)$	
pid	pts_norm	blks_norm	score	rank	score	rank
p1	0.333	1.000	0.500	3	0.833	1
p2	0.667	0.167	0.542	2	0.292	3
q	0.725	0.400	0.644	1	0.481	2

Table 1.1: *Top-k query example.* Shown is a dataset $\mathcal{D}$ of two fictitious basketball player tuples, $p1$ and $p2$, with two normalized attributes, points (pts_norm) and blocks (blks_norm). Also shown is an additional query tuple, $q = (0.725, 0.400)$. Two top-k queries are given in the rightmost columns, along with each tuple's score and rank.

namely $a = (0.75, 0.25)$ and $b = (0.25, 0.75)$. Of these, only query a would be in the response to a *reverse top*-1 query on tuple q, because tuple q is only ranked among the best 1 tuples for query a, not for query b. Both queries would be in the response to a *reverse top*-2 (or top-3) query on q, because tuple q is ranked among the best 2 tuples for both queries.

However, users can specify $\text{top-}k$ queries from the entirety of the line $1 - x, x \in [0, 1]$; so, monochromatic reverse $\text{top-}k$ query solutions are infinite sets. For the dataset in Table 1.1, the reverse top-1 query for the point with qid=1 reports all traditional queries within $[(1.00, 0.00), (0.605, 0.395)]$, a range within which query a, but not query b, falls. We focus on the two-dimensional case and the positive quadrant. We do note, however, that the ideas we present generalize cleanly to all four quadrants, an extension shown to be of significant interest by Ranu and Singh [38].

1.2.1 State of the art

Monochromatic reverse $\text{top-}k$ (mRTOP) queries are quite new and an example of the growing field of *reverse data management* [35]. As yet, there are two algorithms to answer mRTOP queries, the one originally proposed by Vlachou et al. [46], and a subsequent algorithm proposed by Wang et al. [50]. Both are linear-cost, two-dimensional algorithms. Both also have a common limitation, that their computation is heavily centred on and sensitive to the particular query tuple.

Pareto-dominance algorithm [46]

The algorithm of Vlachou et al. [46], refined recently [47], is a two-phase algorithm that utilizes, first, pareto-dominance and, second, a geometric plane sweep. The first step is to scan through $\mathcal{D}$, classifying tuples by pareto-dominance, i.e., into those that dominate q, are dominated by q, and are incomparable to q. A tuple t_i *dominates* another tuple t_j iff

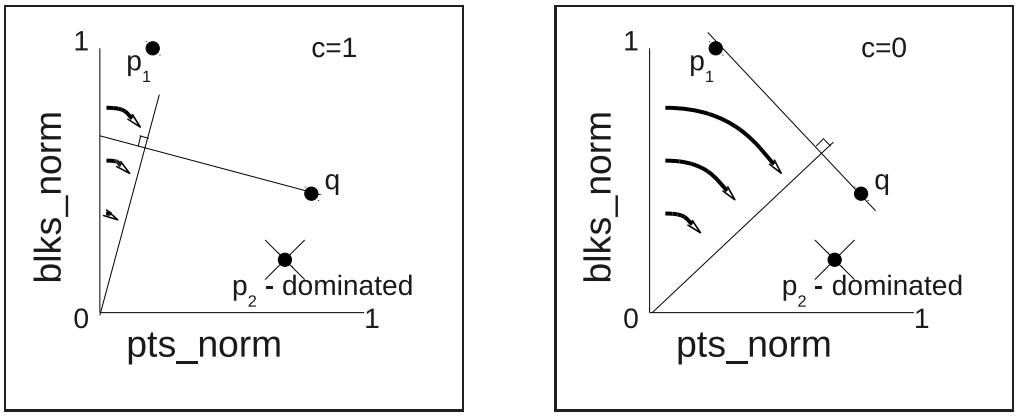


Figure 1.3: *Illustration of the pareto-dominance mRTOP algorithm.* Each subfigure illustrates a different moment of the plane sweep: on the left, point p_1 still outranks q; on the right, q becomes higher ranked than p_1. Point p_2 is dominated by q, so does not affect the plane sweep.

t_i has a value $\geq$ than t_j on every attribute. If neither tuple dominates the other, they are said to be *incomparable*. This produces three sets, respectively: those tuples that always outrank q, never affect the rank of q, or outrank q only for *some* traditional queries.

The second phase models the incomparable tuples as Euclidean points and performs a radial plane sweep that is illustrated in Figure 1.3. At any given moment of the sweep, the number of points, call it c, that outrank q is maintained. Whenever the ranks alternate between q and another tuple, c is updated and the previous angular range is reported if $c < k$. In Figure 1.3, the left subfigure illustrates a moment when p_1 is higher ranked than q. In the right subfigure, the ranks alternate and q becomes the highest ranked tuple. The reverse top-1 response will be the angle of the sweep line at which that occurs, given by the weights $(0.605, 0.395)$, until the termination of the algorithm at the x-axis.

The algorithm inherently depends on knowledge of q and its performance is largely subject to the number of tuples that are incomparable to q. These are limitations that we directly address with our algorithm.

Segmentation algorithm [50]

Wang et al. [50] offer an alternative algorithm based on geometric duality and report an order of magnitude improvement over the pareto-dominance algorithm. Their algorithm requires only a single pass over $\mathcal{D}$ and operates in dual space using the duality transform of Das et al. [15] to convert tuples into lines.

First, they transform the query tuple q into its dual line, l_q. Next, for each tuple $t \in \mathcal{D}$,

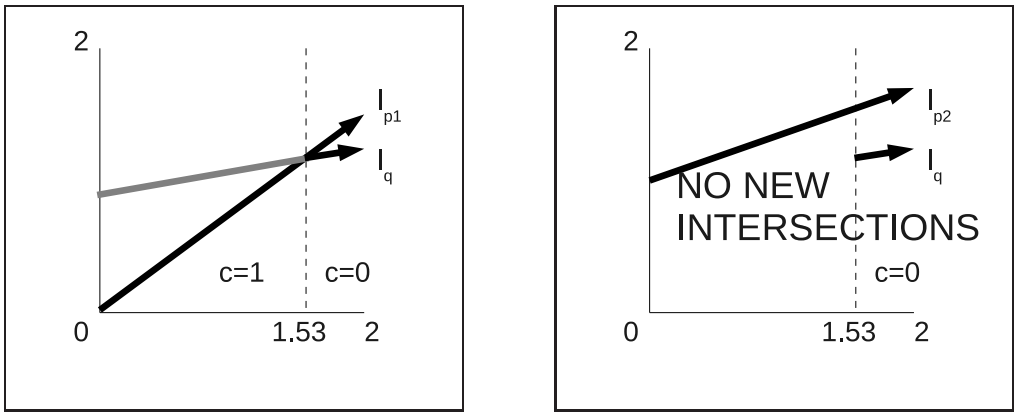


Figure 1.4: *Illustration of the segmentation mRTOP algorithm.* Each (a_1, a_2) is transformed to the line $y = (1-a_1)x+(1-a_2)$ and each traditional top-k query (w_1, w_2) is transformed to $(w_1/w_2, 0)$. The left subfigure illustrates the effect of first processing p_1 and the right subfigure, of subsequently processing p_2.

the dual line l_t is constructed and l_q is fragmented at its intersection point with l_t, as illustrated in Fig. 1.4. Each segment of l_q maintains a counter and this counter is incremented for each segment of l_q that lies above the corresponding segment of l_t. If the counter exceeds $k-1$, then the segment is permanently discarded (as is the case with the leftmost segment in Fig. 1.4). After every tuple in $\mathcal{D}$ has been processed in this manner, the remaining segments together comprise the complete solution to the mRTOP query and are reported (the rightmost segment in Fig. 1.4).

Like the algorithm of Vlachou et al. [46], all processing is inherently dependent on and sensitive to the particular query tuple. The algorithm has the additional disadvantage that it is highly sensitive to the order in which the tuples of $\mathcal{D}$ are traversed, because it is advantageous to discard segments of l_q as quickly as possible. We note that the authors also propose another solution, a rudimentary index that pre-computes the solution to every possible query. But this solution cannot handle the most interesting case of when $q \notin \mathcal{D}$ and it is of limited practical use because of its cubic space complexity.

1.2.2 Our query-agnostic, index approach

Both these approaches suffer from a common limitation: *query-dependence*. Notice in Fig. 1.3 that every step of the plane sweep evaluates whether points lie above or below a line through the point q and perpendicular to the sweep line. Notice in Fig. 1.4 that every step of the dataset traversal compares whether a line l_t lies above or below l_q for numerous (potentially disjoint) intervals. So, in either case, any computation done towards

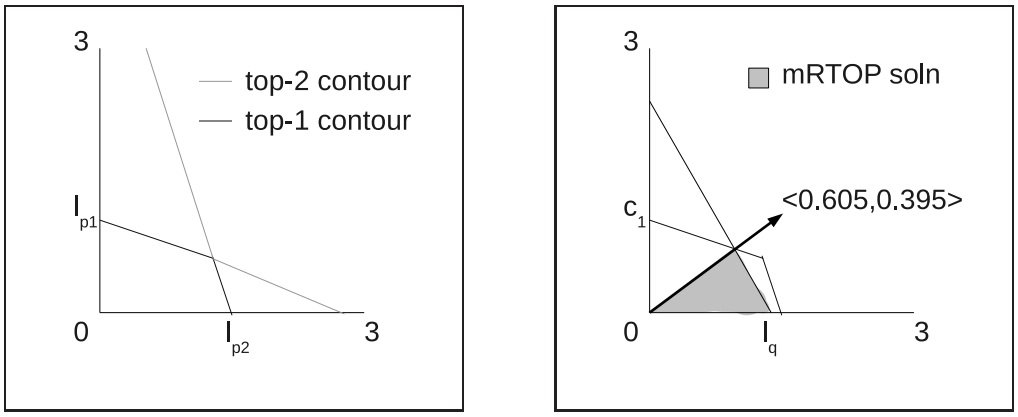


Figure 1.5: *Illustration of our query-agnostic mRTOP algorithm.* To the left, $\mathcal{D}$ is converted to an arrangement of lines by transforming each tuple $t = (a_1, a_2)$ into a line $a_1x+a_2y = \tau$, for an arbitrary, constant real, τ (here, $\tau = 1$). The two contours are shown in different shades. To the right, the first contour is used to answer the reverse top-1 query on q.

resolving one query is unusable for subsequent queries. The computation, involving in both algorithms a full table-scan, must be restarted from scratch. Furthermore, it cannot begin until the query is known.

Our approach to the mRTOP problem, illustrated in Fig. 1.5, is to create an index on $\mathcal{D}$. We do this by employing *four* key geometric techniques: duality, arrangements of lines, plane sweep, and data depth contours, which we will demonstrate in Chapter 3.

Our index, constructed *without knowledge of* q, can respond to many queries, each with only logarithmic cost. The general idea, shown in Fig. 1.5 (left), is to convert $\mathcal{D}$ into an arrangement of lines and identify a critical k-polygon with a plane sweep algorithm. The index is a succinct representation of the polygon. Each query, we prove in Theorem 5, is equivalent to identifying the intersection of a query line l_q with the k-contour, as illustrated in Fig. 1.5 (right).

The outcome of this work is two-fold. For one, we define the top-k rank contour, an exact encoding of the k'th ranked tuples for any possible LOQ (Section 3.2). This is, in fact, a dataset summarization as well. On the other hand, we have an extremely effective technique for pre-processing and later querying monochromatic reverse top-k queries in two dimensions, based on that contour (Section 3.3), that performs quite well on experiments (Section 3.4). So, this chapter focuses on supporting preference queries, for a mRTOP query is about assessing the relevance to preference queries of a given input tuple. But also, this chapter focuses on summarization by defining and studying top-k rank contours.

id	player name	points	rebs	steals	fouls
1	Kevin Durant	**2472**	623	112	171
2	LeBron James	2258	554	125	119
3	Dwyane Wade	2045	373	**142**	181
4	Dirk Nowitzki	2027	520	70	208
5	Kobe Bryant	1970	391	113	187
6	Carmelo Anthony	1943	454	88	225
7	Amare Stoudemire	1896	732	52	**281**
8	Zach Randolph	1681	**950**	80	226

Table 1.2: $\mathcal{D}_{\text{nba}}$. Statistics for the top NBA point scorers from the 2009 regular season, courtesy `databasebasketball.com`. The top score in each statistic is bold.

1.3 k-Regret Minimizing Sets

As we have mentioned, for a user navigating a large dataset, the availability of a succinct representative (i.e., a particularly small) subset of the data points is crucial. For example, consider Table 1.2, $\mathcal{D}_{\text{nba}}$, a toy, but real, dataset consisting of the top eight scoring NBA players from the 2009 basketball season. A user viewing this data might be curious which of these players were "top of the class." That is, he is curious which few points best represent the entire dataset, without his having to peruse it in entirety.

A well-established approach to representing a dataset is with the *skyline* operator [6] which returns all pareto-optimal points. A pareto-optimal point is one for which no other point is higher ranked with respect to every attribute. The skyline operator reduces the dataset down to only those points that are guaranteed to best suit the preferences or interests of *somebody*. If the toy dataset in Table 1.2 consisted only of the attributes *points* and *rebounds*, then the skyline would consist only of the players Kevin Durant, Amare Stoudemire, and Zach Randolph. So, these three players would represent well what are the most impressive combinations of point-scoring and rebounding statistics. However, the skyline is a powerful summary operator only on low dimensional datasets. Even for this toy example, *everybody* is in the skyline if we consider all four attributes. In general, there is no guarantee that the skyline is an especially succinct representation of a dataset. For data either in high dimensions or in anti-correlation, it is rather unlikely that it will be.

1.3.1 Regret minimizing sets

A promising new alternative is the *regret minimizing set*, introduced by Nanongkai et al. [37], which hybridizes the skyline operator with LOQs. A LOQ (or top-k query, as we will often

also call it), recall, takes as input a weight vector **w** and scores each point by inner product with **w**, reporting the k points with highest scores. For example, on weights $\langle .5, .5, 0, 0\rangle$, Randolph earns the highest normalized score ($1681/2472 * .5 + 950/950 * .5 = 0.840$), compared to Kevin Durant next (0.828) and then Kobe Bryant (0.604). So the top-2 query returns Randolph and Durant.

To evaluate whether a subset effectively represents the entire dataset well, Nanongkai et al. introduce *regret ratio* as the ratio of how far from the best score in the dataset is the best score in that subset. For $S = \{\text{Bryant}, \text{Durant}\}$, the regret ratio on a top-1 query $\langle .5, .5, 0, 0\rangle$ is:

$$(0.840 - 0.828)/0.840 = 0.0143,$$

since the score for Randolph is the best in the dataset at 0.840, and the score for Durant is the best in the subset at 0.828. Hence, a user would be 98.57% happy if executing that top-1 query on S rather than all of $\mathcal{D}_{\text{nba}}$.

Motivated to derive a succinct representation of a dataset, one with fixed cardinality, Nanongkai et al. introduce *regret minimizing sets* [37], posing the question, "Does there exist one set of r points that makes every user at least $x\%$ happy (i.e., returns within $x\%$ of correct on any top-1 query)?" In [37], the authors refer to this as a k-regret minimizing set, but we instead refer to this concept as a 1-regret minimizing set of size k, because this term is more natural, especially in the context of our generalization.

For a visual context, consider Fig. 1.6 which depicts all the possible scores for Bryant, Durant, and Randolph on any unit weight vector. The circles are drawn by projecting a vector ending at the data point in all possible directions (relating to Chapter 2). A regret minimizing set R with regret ratio x is the one for which the union of all its circles is within $x\%$ of the outermost circle in any direction (in the positive quadrant). Of the eight basketball players in $\mathcal{D}_{\text{nba}}$, *Zach Randolph* best achieves this criterion, because at worst (the black line on the y-axis) he is closer to Durant's circle than vice versa (the black line on the x-axis). So, Randolph is the regret minimizing set of size 1.

Randolph, however, is a peculiar choice to represent all of $\mathcal{D}_{\text{nba}}$, since he is the worst rated with respect to points. This exposes a weakness of regret minimizing sets: they are forced to fit every outlier in order to satisfy a very rigid criterion for user happiness, that a "happy" user is one who obtains his absolute top choice. However, for an analyst curious to know who is a high rebounding basketball player, is he really unhappy with the second choice, Amare Stoudemire, as a query response rather than Randolph?

In practice, one often does not get the theoretical top choice, anyway. To change the

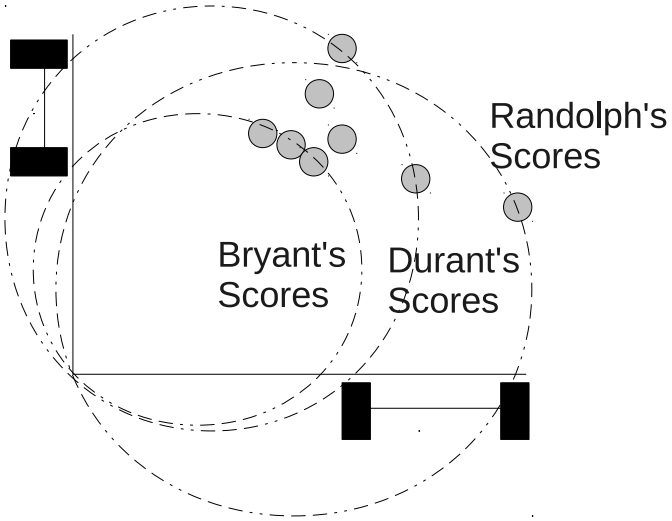


Figure 1.6: *Illustration of regret minimizing sets.* Shown are the possible scores for Durant, Bryant, and Randolph on any unit vector for *rebounds* as the x-attribute and *points* as the y-attribute. The black lines show the maximum distance from Randolph's circle to Durant's, and vice versa, illustrating that Randolph (with the shorter line) is the best single-point approximation to $\mathcal{D}_{\text{nba}}$. The other 5 circles are not shown.

scenario a bit, consider a dataset of hotels and a user searching for one that suits his preferences. The absolute top theoretical choice may not suit him especially well at all. It could be fully booked. Or, he might recall that the manager reminds him of his ex-wife. For a user like him, the regret minimizing set is rigidly constructed on an intangibly poor choice, even if that choice was theoretically far superior.

To alleviate these problems, we soften the happiness criterion to a second or third or fourth "best" point, smoothing out the outliers in the dataset. As a result, with eight points (from the entire dataset of NBA basketball players throughout history, not just the eight players in Table 1.6), we can be within 10% of everyone's third choice, but only within 30% of everyone's top choice. By defining this *k-regret minimizing set* (in Section 4.1), we can more succinctly represent the entire dataset than just with 1-regret. Throughout Chapter 4, we introduce this generalized problem more formally and focus on computational issues around it: showing first that the problem is NP-Hard (Section 4.2), and second that we can design an efficient algorithm in the special case of two dimensions (Section 4.3), using the insight from the previous chapter. Also, we can design an effective albeit inexact algorithm for the general case (Section 4.4), which we evaluate empirically (Section 4.5).

Chapter 2

Threshold Projection Queries

A common trend today is towards designing database queries that are specific to each user. We introduce in this chapter a new class of numeric queries, *threshold projection queries*, for this purpose. We represent every tuple in the database as a vector and return, for a specified user query vector and threshold τ, those database vectors which have a projection onto the query of magnitude at least τ. A primary advantage of these queries is that, contrary to the alternative based instead on *vector dot product*, projection queries have a built-in resilience to bias.

Additionally to introducing the class of queries, this chapter introduces algorithms for indexing numeric datasets to efficiently support threshold projection queries. By employing a duality transform, we construct a general dimension index with worst-case sub-linear query cost. We improve upon this performance for the special case of fixed τ and 2 or 3 dimensions by employing stereographic projection. This yields indices that support queries with logarithmic and square-root I/O cost. The derivation of these algorithms results from the novel geometric insight that is presented in this chapter, the concept of a data vector's *cap*.

Summary of Chapter 2 Contributions

In this chapter we introduce the novel threshold projection queries (TPQs) and are the first to offer database indices to support efficiently responding to them. Specifically, we:

- Cast the indexing problem into a geometry context and derive the novel geometric insight of a vector's *cap* (the reverse mapping from solution to query), crucial to indexing the tuples effectively (Section 2.2).

Symbol	Definition
$\mathcal{D}$	The input, a set of vectors/points/tuples
n	$\|\mathcal{D}\|$, the number of tuples in $\mathcal{D}$
d	The dimension of the problem; # of attributes
v	A tuple in $\mathcal{D}$
$\vec{\mathrm{v}}$	The same tuple $v \in \mathcal{D}$, but as a vector
$\|\|\vec{\mathrm{v}}\|\|$	The magnitude of vector $\vec{\mathrm{v}}$
$\widehat{\mathrm{v}}$	The *cap* of vector $\vec{\mathrm{v}}$
$\vec{\pi}_{\vec{\mathrm{u}}}(\vec{\mathrm{v}})$	The projection of vector $\vec{\mathrm{v}}$ onto vector $\vec{\mathrm{u}}$
τ	A threshold for the size of an 'interesting' projection
$\mathbf{q}$	A query vector, specifying weights for each attribute
$\bar{\mathrm{v}}$	The *baseplane* of vector $\vec{\mathrm{v}}$
s	The size in bytes of a tuple
t	The number of tuples output by a query
b	Average I/O blocksize in bytes

Table 2.1: Table of repeatedly used notation for Chapter 2

- Using a duality transform, produce an indexing algorithm for any dimension $d \geq 2$, utilising the $\mathcal{O}(ns/b)$ simplicial partition tree data structure [1], where s/b reflects the number of blocks occupied by each vector. The query cost of this index is $\mathcal{O}(n^{1-1/d+\epsilon} + ts/b)$ I/O's, where ϵ is any small constant and ts/b reflects the size of the output (Section 2.3).

- Using stereographic projection, produce an alternative indexing algorithm that markedly improves the query bounds for the special case of fixed τ and two or three dimensions. We make use of the interval tree [4] to improve query cost to $\mathcal{O}(lg\ n + ts/b)$ in two dimensions and the priority r-tree [3] for three dimensions to improve query cost to $\mathcal{O}(\sqrt{ns/b} + ts/b)$ I/Os.

2.1 Preliminaries

In this chapter, we study the problem of efficiently retrieving from a set of vectors those that have a sufficiently large projection onto an arbitrary query vector. As argued in Section 1.1, in many contexts this offers a more sensible and less biased form of preference querying. Throughout the chapter, we adopt the convention that vectors have superscript arrows (e.g., $\vec{\mathrm{v}}$), and that the magnitude of a vector $\vec{\mathrm{v}}$ is denoted $||\vec{\mathrm{v}}||$. First recall that the *projection* of a vector $\vec{\mathrm{v}}$ onto another vector $\vec{\mathrm{u}}$ is the component of $\vec{\mathrm{v}}$ in the direction of $\vec{\mathrm{u}}$. More formally:

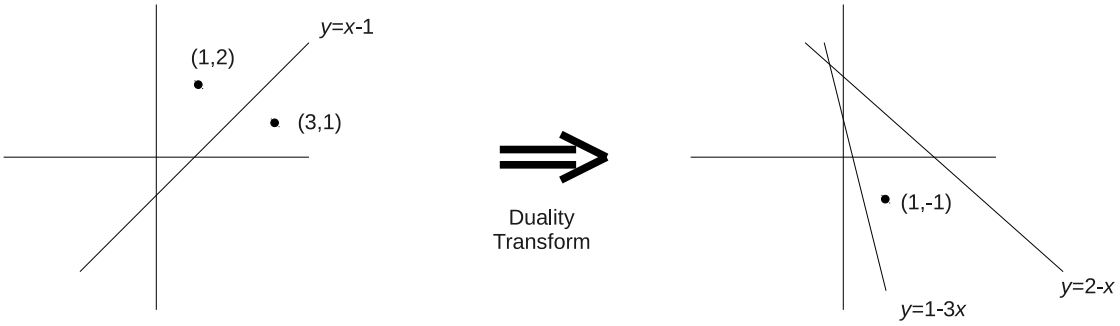


Figure 2.1: *Example of the duality transform*. Throughout this chapter, a point $(1, 2)$ becomes the line $(y = 2 - x)$, the point $(3, 1)$ becomes the line $(y = 1 - 3x)$ and the line $(y = x - 1)$ becomes the point $(1, -1)$. Notice how order is preserved with respect to the origin.

Definition 2.1.1 (Projection)**.** *The projection $\vec{\pi}_{\vec{\mathrm{u}}}(\vec{\mathrm{v}})$ of a vector $\vec{\mathrm{v}}$ onto another vector $\vec{\mathrm{u}}$ is the component of $\vec{\mathrm{v}}$ in the direction of $\vec{\mathrm{u}}$, given by $\left(\frac{\vec{\mathrm{v}}\cdot\vec{\mathrm{u}}}{||\vec{\mathrm{u}}||^2}\right)\vec{\mathrm{u}}$.*

The objective in this chapter is to respond *quickly* to *threshold projection queries* (TPQs). Formally, these queries are defined as follows:

Definition 2.1.2 (Threshold Projection Query (TPQ))**.** *Given a set $\mathcal{D}$ of vectors $\vec{\mathrm{v}} \in \mathbb{R}^d$ and a threshold $\tau \in \mathbb{R}^+$, the result of a* threshold projection query (TPQ) for query vector $\vec{\mathrm{q}} \in \mathbb{R}^d$ is the set $\{\vec{\mathrm{v}} \in \mathcal{D} : ||\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}})|| \geq \tau\}$.

Later in this chapter we apply a duality transform in order to produce an efficient indexing scheme to resolve TPQ queries in arbitrary dimension. A duality transform replaces points (hyperplanes) with hyperplanes (points) in the same-dimensional Euclidean space, preserving both incidence and order. There are many such transforms, so to be specific, we use the definition given below, which is illustrated in Figure 2.1:

Definition 2.1.3 (Duality Transform)**.** *An initial point $p = (a_1, \ldots, a_d)$ or hyperplane $h = (b_d x_d = b_1 x_1 + \ldots + b_{d-1} x_{d-1} + c)$ is referred to as "primal." The duality transform transforms p into its "dual" hyperplane $p^* = (x_d = a_d - a_1 x_1 - \ldots - a_{d-1} x_{d-1})$ and transforms h into its dual point $h^* = (\frac{b_1}{b_d}, \ldots, \frac{b_{d-1}}{b_d}, \frac{c}{b_d})$.*

A critical property of this duality transform is that if a point p lies on the opposite side of a hyperplane h as the origin (i.e., p is *above* h), then h^* is above p^*.

Additionally, *halfspace range searching* (alternatively known as *halfspace range reporting*) is critical to this chapter, because we provide a transformation of our problem into

id	x	y	baseplane	dual point to index
$\vec{a}$	1	2	$y = 1 - \frac{1}{2}x$	$\left(-\frac{1}{2}, 1\right)$
$\vec{b}$	3	2	$y = 1 - \frac{3}{2}x$	$\left(-\frac{3}{2}, 1\right)$
$\vec{c}$	-1	3	$y = \frac{2}{3} + \frac{1}{3}x$	$\left(\frac{1}{3}, \frac{2}{3}\right)$

Table 2.2: *Small example relation to illustrate geometric transformations*. Here, $\tau = 2$.

an instance of halfspace range searching. Given a set of points and a query halfspace, the response to a halfspace range search is the set of points in the query halfspace. Formally:

Definition 2.1.4 (Halfspace range search)**.** *Given a set $\mathcal{D}$ of points in $\mathbb{R}^d$ and a halfspace h, the result of a halfspace range search is the set $\{p \in \mathcal{D} : p \in h\}$.*

For an alternative index, we employ stereographic projection to reduce the dimensionality of the problem. Stereographic projection maps every point on the sphere to a unique point on a plane. The image of a point p under stereographic projection is found by tracing a straight line from p to a pole on the sphere and detecting where that line intersects the plane of projection. In this chapter, we use only the unit sphere and take the pole to be $(0, \ldots, 0, 1)$ and the projection plane to be $x_d = 0$. Thus, the image of p is defined by:

Definition 2.1.5. *The stereographic projection of a point $p = (p_1, \ldots, p_d)$ onto the plane $x_d = 0$ is the point:*

$$\left(\frac{p_1}{1 + p_d}, \ldots, \frac{p_{d-1}}{1 + p_d}, 0\right).$$

2.2 Projection, Caps, and Baseplanes

As mentioned in Section 1.1.3, effectively indexing vectors for TPQ queries is not trivial. So, we introduce the approach of instead indexing *caps*. In this section we formally introduce caps and some important related concepts in order to support the index structures we propose in Sections 2.3 and 2.4.

2.2.1 The Cap of a Vector

Indexing a vector $\vec{\mathrm{v}}$ in its native form is not a promising approach, but as we show, it is very effective to instead construct a representation of all queries for which $\vec{\mathrm{v}}$ should be returned. Because, as argued in Section 1.1.1, $\vec{\mathrm{q}}$ is a unit vector, $\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}}) = \vec{\mathrm{v}} \cdot \vec{\mathrm{q}}$. So, the queries for which $\vec{\mathrm{v}}$ should be returned satisfy two conditions: 1) they lie on the unit sphere (since we assume all queries are of unit length); and 2) they lie within the half-space given by

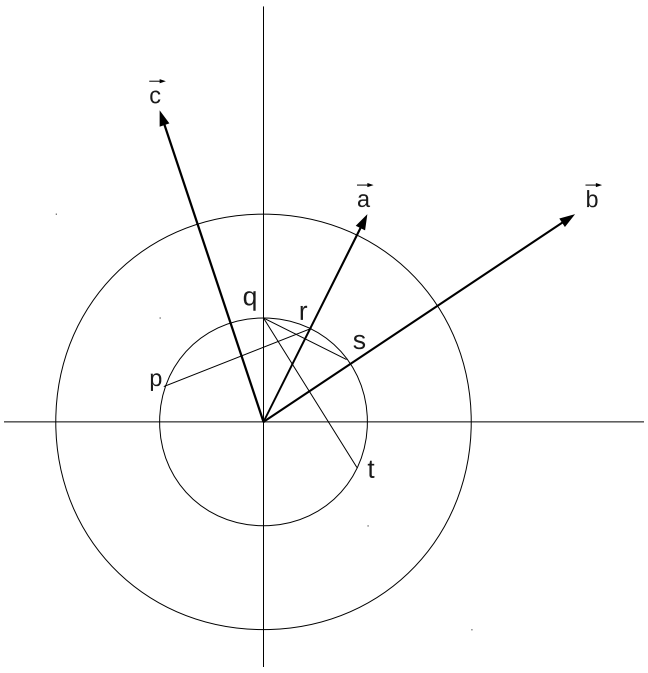


Figure 2.2: *The caps corresponding to the vectors from Table 2.2.* Again, $\tau = 2$. The inner circle is the space of all possible queries, and the outer circle are the vectors of size 2. The caps of $\vec{a}$, $\vec{b}$, and $\vec{c}$ are the arcs $[\mathrm{q}, \mathrm{s}]$, $[\mathrm{q}, \mathrm{t}]$, and $[\mathrm{p}, \mathrm{r}]$, respectively.

$\vec{\mathrm{v}} \cdot \vec{\mathrm{q}} \geq \tau$. This subset is a contiguous geometric object corresponding to the intersection of the surface of the sphere with a halfspace delimited by a hyperplane. Because of its shape in three dimensions (visualise the result of using a cleaver on a hollow pumpkin), we call this object the *cap* of $\vec{\mathrm{v}}$ and denote it $\bar{\mathrm{v}}$. For the example relation of Table 2.2, the cap of each vector is illustrated in Figure 2.2.

The halfspace is delimited by a hyperplane, and the hyperplane is of particular utility here, so we call it the *baseplane* of $\vec{\mathrm{v}}$ and denote it $\bar{\mathrm{v}}$. It is defined to be the unique hyperplane passing through the sphere at points given by query vectors onto which the projection of $\vec{\mathrm{v}}$ is exactly τ. The other geometric object of especial relevance is the component of the baseplane bounded by the unit sphere. In two dimensions, this is a chord (which we call *the chord of* $\bar{\mathrm{v}}$) and in three dimensions it is a disk enclosed by a small circle (which we call *the small circle of* $\bar{\mathrm{v}}$).

Conveniently, the specifications of a cap can be computed quite readily, regardless of the dimension, because the cap is symmetric about the vector. Thus, we can make deductions by observing a planar cross-section of the unit sphere. The next lemma gives these specifications (see Figure 2.3).

Lemma 2.2.1. *The distance from the origin of the unit sphere to the base of the query cap of*

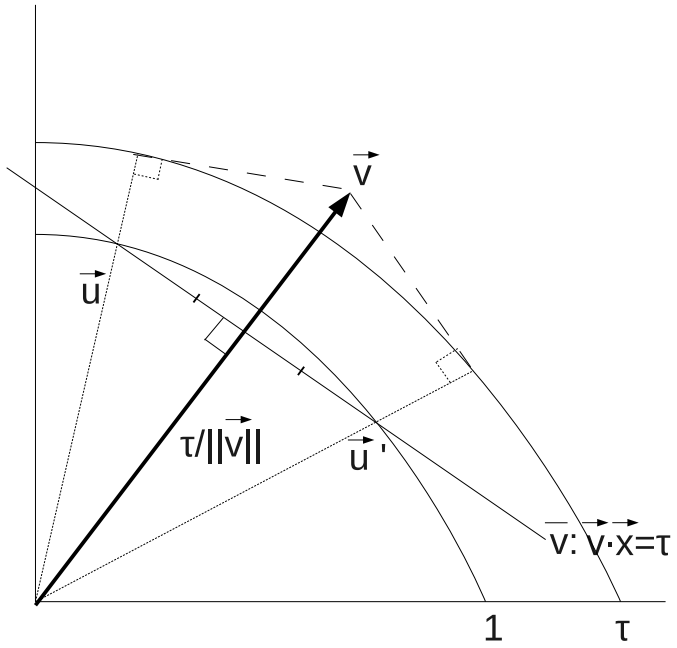


Figure 2.3: *The specifications of a* cap. The positive quadrant of an axis-parallel planar cross section through the origin of a cap. The inner arc is the unit sphere, the set of all possible queries. The outer arc consists of vectors of length τ. The cap is the portion of the unit sphere bounded by the baseplane $\bar{\mathrm{v}}$ orthogonal to $\vec{\mathrm{v}}$ and at a distance of $\frac{\tau}{||\vec{\mathrm{v}}||}$ from the origin. The projection of $\vec{\mathrm{v}}$ onto any vector on this arc is of size at least τ. The unit vectors $\vec{\mathrm{u}}$ and $\vec{\mathrm{u}}$' are the unique pair onto which $\vec{\pi}(\vec{\mathrm{v}})$ is exactly τ. The chord of $\bar{\mathrm{v}}$ is subtended between the vectors $\vec{\mathrm{u}}$ and $\vec{\mathrm{u}}$'.

a vector $\vec{\mathrm{v}} = \langle v_1, \ldots, v_d \rangle$ *is* $\tau/||\vec{\mathrm{v}}||$*, the radius* r *of the query cap is* $\sqrt{(1-\tau/||\vec{\mathrm{v}}||)(1+\tau/||\vec{\mathrm{v}}||)}$*, and the equation of the baseplane is* $\bar{\mathrm{v}} = (v_1 x_1 + \ldots + v_d x_d - \tau = 0)$.

Proof Consider some unit vector $\vec{\mathrm{u}}$ such that $\vec{\pi}_{\vec{\mathrm{u}}}(\vec{\mathrm{v}}) = \tau$. Together, $\vec{\mathrm{u}}$ and $\vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}})$ create a triangle with the line segment r that joins their endpoints. Because $\vec{\mathrm{u}}$ is of unit length, $\vec{\mathrm{u}} \cdot \vec{\mathrm{v}} = \vec{\pi}_{\vec{\mathrm{u}}}(\vec{\mathrm{v}}) = \tau$. Thus, $\vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}}) = \frac{\vec{\mathrm{u}} \cdot \vec{\mathrm{v}}}{||\vec{\mathrm{v}}||} = \frac{\tau}{||\vec{\mathrm{v}}||}$. Additionally, from the Pythagorean Theorem,

$$r = \sqrt{\left(1 - \frac{\tau}{||\vec{\mathrm{v}}||}\right)\left(1 + \frac{\tau}{||\vec{\mathrm{v}}||}\right)}.$$

These measurements give, respectively, the distance from the origin and the radius r of the cap.

The plane can be determined in point-plane (Hessian Normal) form in time linear in d. The orthogonal vector is $\vec{\mathrm{v}}$ and a point on $\bar{\mathrm{v}}$ is given by the position vector $||\vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}})||\vec{\mathrm{v}}$. The equation of the plane can be determined by solving the nullspace equation and translating

it to the initial point given by $||\vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}})||\vec{\mathrm{v}}$.

$$\begin{aligned}
\bar{\mathrm{v}} \Leftrightarrow \quad & (\vec{\mathrm{v}} \cdot (\vec{\mathrm{x}} - \vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}})\vec{\mathrm{v}}) && = 0) \\
\Leftrightarrow \quad & (\vec{\mathrm{v}} \cdot \vec{\mathrm{x}} - \vec{\mathrm{v}} \cdot \vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}})\vec{\mathrm{v}} && = 0) \\
\Leftrightarrow \quad & (\vec{\mathrm{v}} \cdot \vec{\mathrm{x}} - v\vec{\pi}_{\vec{\mathrm{v}}}(\vec{\mathrm{u}}) && = 0) \\
\Leftrightarrow \quad & (v_1 x_1 + \ldots + v_d x_d - \tau && = 0).
\end{aligned}$$

□

By converting every vector into its cap, we can design effective spatial indices for the caps and work within the transformed problem space. Let q denote the endpoint of the query vector $\vec{\mathrm{q}}$. Determining which caps contain q is equivalent to solving the original TPQ problem:

Theorem 1 (Equivalence of TPQ and cap containment)**.**
Given a set of vectors $\mathcal{D}$, a threshold τ, and a query vector $\vec{\mathrm{q}}$, $\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}}) \geq \tau \Leftrightarrow q \in \hat{\mathrm{v}}$.

Proof First, $\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}}) \geq \tau \Rightarrow q \in \hat{\mathrm{v}}$ by construction. To prove $\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}}) \geq \tau \Leftarrow q \in \hat{\mathrm{v}}$, note that $q \in \hat{\mathrm{v}}$ implies that q is on the unit sphere and that it is in the halfplane spanned by vectors $\vec{\mathrm{x}}$ such that $\vec{\mathrm{v}} \cdot \vec{\mathrm{x}} \geq \tau$. So, the unit vector $\vec{\mathrm{q}}$ in the direction of point q is such that $\vec{\pi}_{\vec{\mathrm{q}}}(\vec{\mathrm{v}}) \geq \tau$. □

2.3 An Index for Arbitrary Dimension

The task of indexing caps to solve the cap-containment problem (*in which caps is the query*?) in arbitrary dimension for arbitrary τ is one that can be reformulated in such a way as to take advantage of existing efficient external memory data structures. In particular, we employ a duality transform (Section 2.3.1) and then demonstrate (Section 2.3.2) how a simplicial partition tree can resolve queries with $\mathcal{O}(n^{1-1/d+\epsilon} + ts/b)$ I/O's, for a dataset of size n in d dimensions and any small constant $\epsilon > 0$, with t output vectors each occupying s bytes and a blocksize of b bytes per block of I/O. The value ts/b reflects the number of blocks of output. The data structure requires linear $\mathcal{O}(ns/b)$ space. Sections 2.3.1 and 2.3.2 detail how the data structure is constructed using an arbitrary seed threshold τ. Finally but importantly, in Section 2.3.3 we demonstrate how a geometric shift applied to incoming queries is sufficient to support *any* dynamic (i.e., user-supplied) positive thresholds, not just the static one with which the data structure is built.

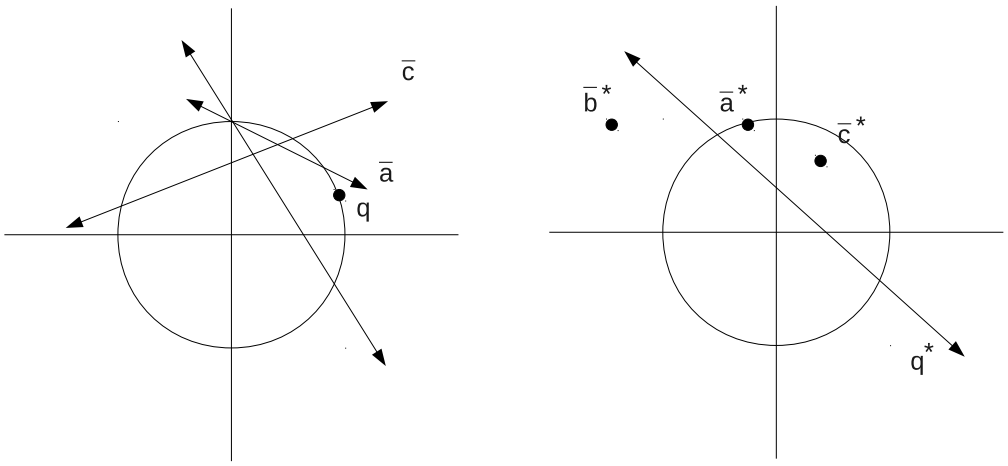


Figure 2.4: *Example of dataset vectors transformed into baseplanes.* Shown are the baseplanes of the vectors in Table 2.2 (left), together with their dual points (right), using $\tau = 2$. Also depicted is a sample query, $\vec{\mathrm{q}} \approx \langle .9, .4 \rangle$ and its dual, the line $y \approx .4 - .9x$. Notice the inversion of aboveness in the dual space.

2.3.1 Venturing into the Dual Space

It is clear from Section 2.2 that a dataset of n vectors can be interpreted as n caps, or, equivalently, as n baseplanes, and a query $\vec{\mathrm{q}}$ can be regarded as a point q. Since a normalised query will always produce a point on the unit sphere, checking whether q lies above a baseplane $\bar{\mathrm{v}}$ is sufficient to determine if $\vec{\mathrm{q}}$ is in $\bar{\mathrm{v}}$. So, the problem is to determine the set of baseplanes above which q lies. It is to this problem that we will apply a duality transform.

Recall from Definition 2.1.3 of a duality transform that it inverts "aboveness." Thus, if one point p is above a particular hyperplane h, then h's dual point h^* will be above the point's dual hyperplane p^*.

We convert each cap into a point by applying the duality transform to its baseplane, thus obtaining a set of n dual points. The position vector of any query can be transformed into a hyperplane. This transforms the problem into a halfspace range search, as described in Proposition 2.3.1.

Proposition 2.3.1 (Equivalence of cap-containment to halfspace range search)**.**
Let $\mathcal{Q}$ denote a set of vector baseplanes and let $p > h$ denote that point p lies on the opposite side of the hyperplane h as does the origin (i.e., is above h). Then, for a given query $\vec{\mathrm{q}}$, $\{h \in \mathcal{Q} : q > h\} = \{h \in \mathcal{Q} : h^* > q^*\}$.

In other words, by applying a duality transform, the problem of determining in which caps a particular query lies becomes a case of halfspace range searching.

Algorithm 1 Indexing for TPQs in general dimension with dynamic τ.

Input: A vector dataset $\mathcal{D}$ and a seed threshold τ
Output: An index supporting sub-linear query time
Create an empty point set $\mathcal{S}$
for all $\vec{\mathrm{v}} = \langle a_1, \ldots a_d \rangle \in \mathcal{D}$ **do**
  Compute the point $\bar{\mathrm{v}}^* = \left(-\frac{a_1}{a_d}, \ldots, -\frac{a_{d-1}}{a_d}, \frac{\tau}{a_d} \right)$
  Add $\bar{\mathrm{v}}^*$ to $\mathcal{S}$
end for
Index the set $\mathcal{S}$ in the external memory simplicial partition tree
Return the external memory simplicial partition tree

Using the particular duality transform given earlier, we transform the baseplane $\bar{\mathrm{v}}$ (namely $x_d = -\frac{v_1}{v_d}x_1 - \ldots - \frac{v_{d-1}}{v_d}x_d + \frac{\tau}{v_d}$) into the dual point $\bar{\mathrm{v}}^* = (-\frac{v_1}{v_d}, \ldots, -\frac{v_{d-1}}{v_d}, \frac{\tau}{v_d})$. Figure 2.4 illustrates the baseplanes and their dual points for the vectors given earlier in Table 2.2.

2.3.2 Constructing the Index

By means of this duality transformation, the threshold projection problem can be reformulated as a case of halfspace range searching. The purpose of this is to take advantage of the extensive research that has already been conducted on the halfspace range searching problem. The external memory simplicial partition tree data structure given by Agarwal et al. [1] requires linear $\mathcal{O}(ns/b)$ space and can answer halfspace range search queries in $\mathcal{O}(n^{1-1/d+\epsilon} + ts/b)$ I/O's.

The series of transformations from a vector to a cap to a dual point can be arithmetically combined into one computation. Thus, as a result of Theorem 1 and Proposition 2.3.1, we have Algorithm 1 for preprocessing a dataset $\mathcal{D}$ into a simplicial partition tree index in order to efficiently respond to threshold projection queries.

Then, for each query $\vec{\mathrm{q}}$, one can compute the dual hyperplane q^* as $(x_d = q_d - q_1 x_1 - \ldots - q_{d-1}x_{d-1})$ in real-time and execute a halfspace range search.

2.3.3 Querying the Index

Until this point, we have held τ fixed in order to construct the data structure. Here we discuss how the orthogonality of the data vector to the baseplane of its cap allows us to efficiently transform the query to respond to new, dynamic thresholds, rather than just the

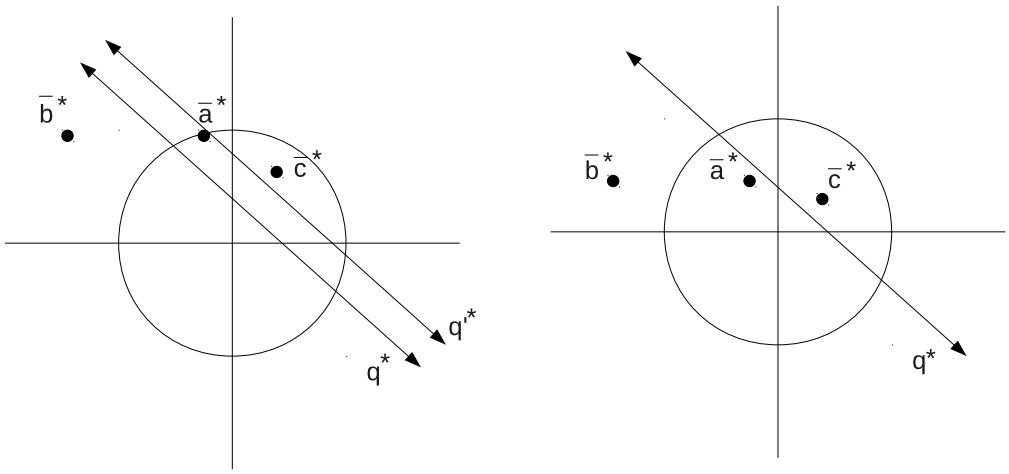


Figure 2.5: *Adjusting caps for different thresholds*. In this illustration, we adjust the example caps from Table 2.2 for a new threshold of $\tau' = 1$ rather than $\tau = 2$. The reduced threshold permits $\vec{a}$ to become part of the result set. To the right, all the points are translated as if the caps had been originally created with a threshold of $\tau' = 1$. To the left, on the other hand, the same result set is achieved by translating only the query line, as per Theorem 2.

seed threshold τ required to initialise the data structure, and, indeed, how to respond to any user query $\vec{\mathrm{q}}$, τ.

Recall from Algorithm 1 that each vector $\vec{\mathrm{v}}$ is transformed into a point $\bar{\mathrm{v}}^*$:

$$\bar{\mathrm{v}}^* = \left(-\frac{v_1}{v_d}, \ldots, -\frac{v_{d-1}}{v_d}, \frac{\tau}{v_d} \right).$$

Consider what happens if $\vec{\mathrm{v}}$ is scaled to $c\vec{\mathrm{v}}$: it is transformed to the new point $c\bar{\mathrm{v}}^*$:

$$\begin{aligned} c\bar{\mathrm{v}}^* &= \left(-\frac{cv_1}{cv_d}, \ldots, -\frac{cv_{d-1}}{cv_d}, \frac{\tau}{cv_d} \right) \\ &= \left(-\frac{v_1}{v_d}, \ldots, -\frac{v_{d-1}}{v_d}, \frac{\tau}{cv_d} \right). \end{aligned}$$

Figure 2.5 illustrates how an entire dataset is transformed in this nature. The direction of the vector is captured by the first d-1 coordinates of the dual point and its magnitude is described by the last coordinate. This is intuitive since the baseplanes of the caps of $\vec{\mathrm{v}}$ and $c\vec{\mathrm{v}}$ are parallel to each other. The sufficiency of the first d-1 coordinates in capturing the direction of the baseplane results from the fact that the nullspace of a line only spans d-1 dimensions and it is from translating the nullspace of $\vec{\mathrm{v}}$ that $\bar{\mathrm{v}}$ is derived.

We exploit this fact as follows. Recall that a query $\vec{\mathrm{q}}$ will be transformed into a dual halfspace $q^* = (x_d = q_d - q_1 x_1 - \ldots - q_{d-1} x_{d-1})$. This is expressed in point-intercept

form for insight: the value of q_d shifts the query up or down the x_d axis. Given that the relationship between the magnitude of a vector $\vec{\mathrm{v}}$ and the threshold τ is expressed along the x_d axis, such a shift effectively changes the threshold. If the user wishes to instead use a threshold of τ', one need only adjust q^* to $\left(x_d = \frac{q_d\tau}{\tau'} - q_1x_1 - \ldots - q_{d-1}x_{d-1}\right)$, and the index remains unaffected. See Figure 2.5 (top).

Theorem 2 (Transformation of τ to τ')**.**
For a vector projection query index initialised with a threshold τ, the response to a query vector $\vec{\mathrm{q}} = \langle q_1, \ldots, q_d\rangle$ for another threshold τ' is the same as the response to query vector $\langle q_1, \ldots, q_{d-1}, q_d\tau/\tau'\rangle$ with threshold τ.

Proof We show this by considering an arbitrary data vector, $\vec{\mathrm{v}} = \langle v_1, \ldots, v_d\rangle$. Its cap is represented by the dual point $(-\frac{a_1}{a_d}, \ldots, -\frac{a_{d-1}}{a_d}, \frac{\tau}{a_d})$ for a given threshold τ and represented by $(-\frac{a_1}{a_d}, \ldots, -\frac{a_{d-1}}{a_d}, \frac{\tau'}{a_d})$ for a given threshold τ'. When the threshold is modified from τ to τ', clearly every cap dual point is moved along the x_d axis by a factor of τ'/τ and the other coordinates remain unchanged.

So, if the query dual halfplane is shifted by the same factor in the opposite direction, then the above-below relationship is preserved. Scaling two real values by the same positive factor cannot alter their order with respect to each other. □

This also suggests how some problem variants, such as the top k variant, can be answered, by shifting the query dual hyperplane up and down the x_d axis until an appropriate output size is obtained.

So what of the original seed τ? First, it is sufficient in a static setting. But in dynamic scenarios, its role is to establish a relationship between the size of a vector and the size of its cap. By choosing τ to be some constant, each cap can be constructed so that it contains a portion of the unit sphere that is proportional to the vector's magnitude (actually to $\sqrt{(1-\tau/||\vec{\mathrm{v}}||)(1+\tau/||\vec{\mathrm{v}}||)}$, to be precise). Some care should be taken in choosing the seed τ, however, because a value that is extremely small or extremely large relative to the domain of the attributes could lead to difficulties with floating point arithmetic.

2.4 Exploiting Low Dimension and Fixed τ

In Section 2.3, we gave an index for arbitrary dimension that permitted dynamic adaptation (i.e., user-specification) of the threshold value τ as any positive, real number. To do so, we employed a data structure whose query cost is sub-linear in n. Yet, one can imagine

settings in which τ either remains static or is confined to a small, finite set of possible values (and thus serviceable by a small, finite set of indices). For example, an interface could be designed to constrain a user to selecting among a few sensible options for τ, rather than allowing him to choose any arbitrary real from within a given window of values. Alternatively, most systems will perform some post-processing before presenting results to a user (even if only formatting), and to post-process a small result set of a smaller than desired τ in order to eliminate false positives would be of negligible cost. In such settings, we can exploit the static nature of τ to produce a data structure with better query cost.

We begin by giving an overview of the indexing algorithm for this setting. In the dynamic setting, we required a transform that permitted adapting the threshold with a quick translation of the query. For this purpose, the duality transform was well suited. With fixed τ, however, that requirement does not exist, so we have more freedom in choosing spatial indices for the caps. We exploit this advantage by using stereographic projection and an interval tree (in 2d), or priority r-tree (in 3d), for which the query cost is logarithmic and $\mathcal{O}(\sqrt{n})$, respectively.

The choice to use stereographic projection (rather than some other projection) on the unit sphere has two very nice consequences as a result of being a conformal mapping: 1) the image of a small circle of the sphere is another circle on the projection plane; and 2) any point within a small circle on the sphere has a corresponding image within the image of the small circle.

Consequently, determining those caps whose image contains the image of the query is sufficient to resolve the TPQ. This forms the basis for our fixed-τ index: we use spatial indices to store the images of the caps. For each query, we compute its image and retrieve from the spatial index those cap images within which the query's image is contained. We now describe each step of the indexing algorithm in greater detail.

2.4.1 The Stereographic Projection of a Cap

The first step is to project caps onto the projection plane ($y = 0$ for $2d$ and $z = 0$ for $3d$). Recall from Definition 2.1.5 that the image of a point $(p_1, \ldots, p_d)$ in the hyperplane $x_d = 0$ is:

$$\left(\frac{p_1}{p_d + 1}, \ldots, \frac{p_{d-1}}{p_d + 1}, 0 \right).$$

Figure 2.6 illustrates the stereographic projection of a two- (left) and a three- (right) dimensional cap. In three dimensions, each cap can be described by a *small circle*, and in two dimensions, by a *chord*. Their images under stereographic projection are, respectively,

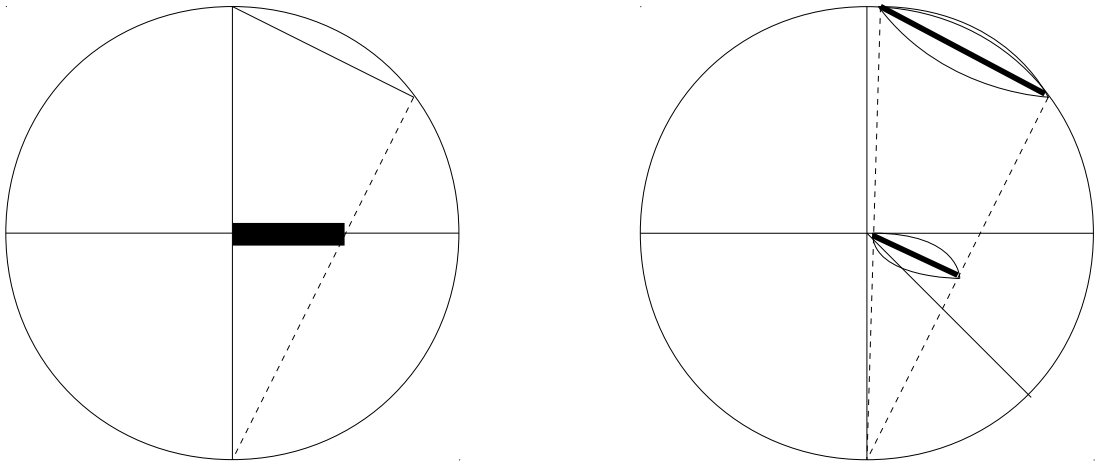


Figure 2.6: *Stereographic projection*. The chord (on the left) and the small circle (on the right) of a cap are stereographically projected onto the x-axis (left) and the xy-plane (right).

a circle and a line segment.

In order to find the image of $\bar{\mathrm{v}}$ in three dimensions, we first find the endpoints of some diameter of $\bar{\mathrm{v}}$'s small circle (the upper dark solid line in Figure 2.6 (right)) and project the diameter's endpoints. These projected points define a line segment in the $z = 0$ plane, call it l (the lower dark solid line). Because the image of $\bar{\mathrm{v}}$ is a circle of which l is a diameter, rotating l through π radians in the $z = 0$ plane produces the image of the entire cap.

In two dimensions, on the other hand, the image of $\bar{\mathrm{v}}$ is simply the line segment without rotation. In either dimension, l can be computed explicitly and in main memory time linear in d, as indicated in Theorem 3.

Theorem 3 (Stereographic images of 2d and 3d caps)**.**
In two dimensions, the stereographic projection of $\bar{\mathrm{v}}$ is the line segment:

$$\left[\left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 + \frac{r}{X}}{Y - \frac{rv_1}{Xv_2}}, 0\right), \left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 - \frac{r}{X}}{Y + \frac{rv_1}{Xv_2}}, 0\right)\right],$$

where:

$$X = \sqrt{1 + \left(\frac{v_1}{v_2}\right)^2}, Y = 1 + \frac{\tau}{||\vec{\mathrm{v}}||^2}v_2, r = \sqrt{1 - \left(\frac{\tau}{||\vec{\mathrm{v}}||}\right)^2}.$$

Similarly, in three dimensions, a diameter of the stereographic projection of $\bar{\mathrm{v}}$ is the

line segment:

$$\left[\left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1}{Y-\frac{rv_2}{Xv_3}},\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_2+\frac{r}{X}}{Y-\frac{rv_2}{Xv_3}},0\right),\left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1}{Y+\frac{rv_2}{Xv_3}},\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_2-\frac{r}{X}}{Y+\frac{rv_2}{Xv_3}},0\right)\right]$$

where:

$$X=\sqrt{1+\left(\frac{v_2}{v_3}\right)^2}, Y=1+\frac{\tau}{||\vec{\mathrm{v}}||^2}v_3, \text{ and } r \text{ is as before.}$$

Proof Recall that to project a cap in $2d$ ($3d$) onto the line (plane) $y=0$ ($z=0$), we need to compute the image of the chord (diametre) that spans the cap. Thus the proof proceeds in two parts: first we find the endpoints of the chord (diametre), and then we compute the image of those endpoints. We begin by outlining the intuition of the proof.

In $2d$, the chord is unique. In $3d$, there are infinitely many diametres of the cap's small circle, and any will suffice. In either case, the intuition is the same: we can compute the endpoints of the chord (diametre) by recognising them as the vector addition of $\vec{\mathrm{v}}$ appropriately scaled and another appropriately scaled vector in the nullspace of $\vec{\mathrm{v}}$. Recall Figure 2.3. Given $\vec{\mathrm{v}}$, it is the endpoints of $\vec{\mathrm{u}}$ and $\vec{\mathrm{u}}'$ that are sought, and they both lie on $\bar{\mathrm{v}}$. We then project those two points onto $x_d=0$ using the mapping:

$$(p_1,\ldots,p_d)\mapsto\left(\frac{p_1}{1+p_d},\ldots,\frac{p_{d-1}}{1+p_d},0\right).$$

The details are as follows, first described in $2d$. Recall from Theorem 2.2.1 that $\bar{\mathrm{v}}$ is at a distance of $\frac{\tau}{||\vec{\mathrm{v}}||}$ from the origin and that the radius of $\bar{\mathrm{v}}$ is $r=\sqrt{1-\left(\frac{\tau}{||\vec{\mathrm{v}}||}\right)^2}$. Let $\vec{\mathrm{v}}'=\frac{\tau}{||\vec{\mathrm{v}}||^2}\vec{\mathrm{v}}$, the vector $\vec{\mathrm{v}}$ scaled to where it intersects its baseplane. Let u and u' be the two sought points, which are at a distance of 1 from the origin and that delimit the chord of $\bar{\mathrm{v}}$. Finally, let η be a unit vector from $\vec{\mathrm{v}}'$ towards u, clearly in the nullspace of $\vec{\mathrm{v}}$. So, u and u' are the two endpoints of the vectors:

$$\{\vec{\mathrm{u}},\vec{\mathrm{u}}'\}=\frac{\tau}{||\vec{\mathrm{v}}||^2}\vec{\mathrm{v}}\pm r\eta.$$

The direction of η can be obtained from solving the nullspace equation of $\vec{\mathrm{v}}$ while fixing

$x_1 = 1$:

$$v_1 + x_2 v_2 = 0 \Rightarrow x_2 = -\frac{v_1}{v_2}.$$

So, $\eta = \frac{1}{\left(\sqrt{1+\left(\frac{v_1}{v_2}\right)^2}\right)} \left\langle 1, -\frac{v_1}{v_2} \right\rangle = \left\langle 1, -\frac{v_1}{v_2} \right\rangle / X.$

Thus the image of the chord of $\hat{\mathrm{v}}$ is given by computing the piecewise addition of the described vectors and then applying stereographic projection:

$$\begin{aligned}
[u', u] &= \left[\frac{\tau}{||\vec{\mathrm{v}}||^2}\vec{\mathrm{v}} \pm r\eta\right] \\
&= \left[\left(\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 + \frac{r}{X}, \frac{\tau}{||\vec{\mathrm{v}}||^2}v_2 - \frac{rv_1}{Xv_2}\right), \left(\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 - \frac{r}{X}, \frac{\tau}{||\vec{\mathrm{v}}||^2}v_2 + \frac{rv_1}{Xv_2}\right)\right] \\
&\mapsto \left[\left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 + \frac{r}{X}}{1 + \frac{\tau}{||\vec{\mathrm{v}}||^2}v_2 - \frac{rv_1}{Xv_2}}, 0\right), \left(\frac{\frac{\tau}{||\vec{\mathrm{v}}||^2}v_1 - \frac{r}{X}}{1 + \frac{\tau}{||\vec{\mathrm{v}}||^2}v_2 + \frac{rv_1}{Xv_2}}, 0\right)\right].
\end{aligned}$$

The three dimensional case is analogous. To compute the direction of η we fix $x_1 = 0$ and $x_2 = 1$ to produce:

$$\eta = \frac{1}{\left(\sqrt{1+\left(\frac{v_2}{v_3}\right)^2}\right)} \left\langle 0, 1, -\frac{v_2}{v_3} \right\rangle = \left\langle 0, 1, -\frac{v_2}{v_3} \right\rangle / X.$$

Then, the image of a diametre of the small circle of $\hat{\mathrm{v}}$ is:[1]

$$\begin{aligned}
[u', u] &= [\vec{\mathrm{v}}' \pm l\eta] \\
&= \left[\left(\vec{\mathrm{v}}'_1, \vec{\mathrm{v}}'_2 + \frac{r}{X}, \vec{\mathrm{v}}'_3 - \frac{rv_2}{Xv_3}\right), \left(\vec{\mathrm{v}}'_1, \vec{\mathrm{v}}'_2 - \frac{r}{X}, \vec{\mathrm{v}}'_3 + \frac{rv_2}{Xv_3}\right)\right] \\
&\mapsto \left[\left(\frac{\vec{\mathrm{v}}'_1}{1 + \vec{\mathrm{v}}'_3 - \frac{rv_2}{Xv_3}}, \frac{\vec{\mathrm{v}}'_2 + \frac{r}{X}}{1 + \vec{\mathrm{v}}'_3 - \frac{rv_2}{Xv_3}}, 0\right), \left(\frac{\vec{\mathrm{v}}'_1}{1 + \vec{\mathrm{v}}'_3 + \frac{rv_2}{Xv_3}}, \frac{\vec{\mathrm{v}}'_2 - \frac{r}{X}}{1 + \vec{\mathrm{v}}'_3 + \frac{rv_2}{Xv_3}}, 0\right)\right].
\end{aligned}$$

□

[1] Note that we express this in terms of $\vec{\mathrm{v}}'$ instead of $\vec{\mathrm{v}}$ for the sake of lateral real-estate on this page.

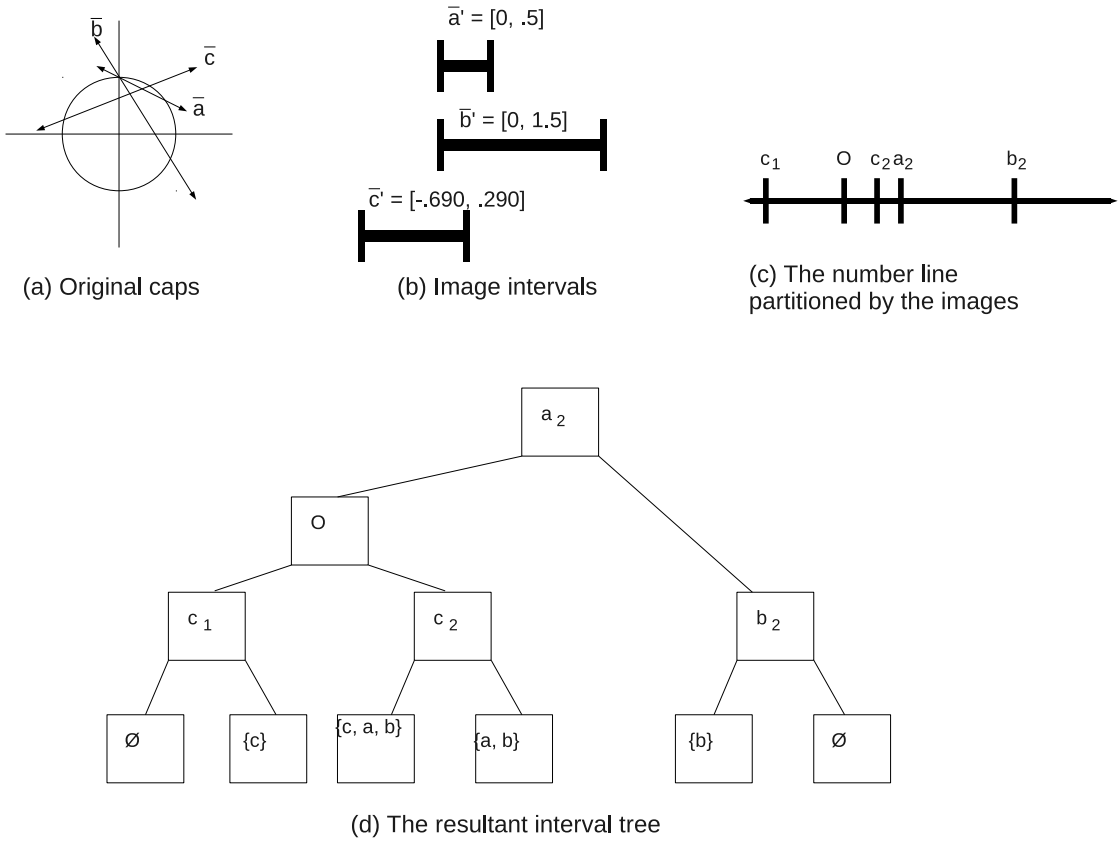


Figure 2.7: *Illustration of* $2d$ *case of Algorithm 2.* These are the four steps in converting the vectors of Table 2.2 into an interval tree data structure. Displayed are the caps (a), the images under stereographic projection of the caps (b), the partitioning of the number line based on the images (c), and the final tree structure (d).

2.4.2 An Index for Fixed τ and 2 or 3 Dimensions

The idea for the fixed-τ index is to use a spatial index to organise the stereographic images of the caps, and query the spatial index with the image of the query. The image of a query $\vec{\mathrm{q}}$ is:

$$\left(\frac{q_1}{1+q_d}, \ldots, \frac{q_{d-1}}{1+q_d}, 0 \right),$$

a straightforward consequence of Definition 2.1.5.

In two dimensions, the image of every cap (arc), is a line segment given by Theorem 3. Also, $\vec{\mathrm{q}}$ is in $\bar{\mathrm{v}}$ iff the image of $\vec{\mathrm{q}}$ (a point) lies on the image of $\bar{\mathrm{v}}$ (a line segment). Stated alternatively, the task of identifying which vectors should be returned for the threshold projection query $\vec{\mathrm{q}}$ is equivalent to identifying all line segments on which the image

Algorithm 2 Indexing for static τ

Input: A vector dataset $\mathcal{D}$ and a threshold τ
Output: An index supporting worst-case logarithmic (2d) or square-root (3d) query time
Create an empty set of line segments $\mathcal{L}$ (2d) or circles $\mathcal{C}$ (3d).
for all $\vec{\mathrm{v}} \in \mathcal{D}$ **do**
  Compute the image of $\vec{\mathrm{v}}$ in constant time, as per Theorem 3
  Add the image of $\vec{\mathrm{v}}$ to $\mathcal{L}$ (2d) or to $\mathcal{C}$ (3d)
end for
Index the set $\mathcal{L}$ in an external memory interval tree (2d) or $\mathcal{C}$ in a Priority r-Tree (3d)
Return the tree

of $\vec{\mathrm{q}}$ lies. In fact, this is precisely known as the Interval Stabbing Problem [4] (*which one-dimensional overlapping intervals does a given point 'stab'*?). The Interval Stabbing Problem is optimally supported in external memory by the interval tree data structure of Arge and Vitter [4], which uses linear $\mathcal{O}(ns/b)$ space, and can respond to queries with optimal $\mathcal{O}(lg\ n + ts/b)$ I/Os. The example relation from Table 2.2 is transformed into an interval tree in Figure 2.7.

In three dimensions, on the other hand, the image of every cap is a circle on the plane, given by rotating the line segment of Theorem 3. This gives rise to a degenerate window query (alt., a *where-am-i* query) in which the query window is a single point. Window queries are handling quite effectively by the several r-tree variants. The *priority r-tree* [3], in particular, is asymptotically optimal and requires $\mathcal{O}(ns/b)$ disk space. It guarantees worst-case query cost of $\mathcal{O}(\sqrt{ns/b} + ts/b)$ I/Os.

To summarise, we have Algorithm 2 for indexing two- and three-dimensional vectors for TPQ with fixed τ.

2.5 Bibliographic Notes

Threshold projection queries are a new concept, introduced in this work. A condensed version of the material in this chapter originally appeared at DASFAA 2011 [10] and an expanded version is currently under revision for a top-tier international journal [14].

The closest problem to the one under study here is that of indexing for the top-k linear optimisation queries (LOQs) that we discussed in Section 1.1 (c.f., [1, 9, 20, 34, 53, 57]). Tsaparas et al. [44] design indices for more general monotone functions; however, their techniques are only for top-k and problems in the $2d$ plane.

Our work leveraged several techniques and existing data structures. In Section 2.3,

we utilised a duality transform, which allowed us to cast the problem into an instance of halfspace range searching. For more information about duality transforms we refer to the Computational Geometry text of de Berg et al. [5] and to the survey by Matoušek [33], both which demonstrate their use nicely and repeatedly. Range searching is a canonical problem in Computational Geometry, of which halfspace range searching is a specific, well-studied case. In higher dimensions, the asymptotically best algorithm is the one due to Matoušek [32], which supports queries in $\mathcal{O}(n^{1-\frac{1}{d}+\epsilon})$ time, for any fixed constant $\epsilon > 0$, and uses linear space for the data structure. The external memory simplicial partition tree of Agarwal et al. [1] is an adaptation for disk of Matoušek's data structure.

For the fixed-τ setting in Section 2.4, we use stereographic projection rather than a duality transform. For a nice introduction to stereographic projection, we refer to Whittaker [51]. In the two-dimensional case, we employ the optimal interval tree data structure of Arge and Vitter [4]. In three dimensions, we employ the optimal priority r-tree data structure of Arge et al. [3].

It is worth noting that the general problem with which we are faced is to index small circles of a (d-1)-sphere. In three dimensions, this problem is of particular interest, because it relates strongly to indexing objects on the Earth (for GIS systems) and on the celestial sphere (for astronomical applications). Within this domain, another approach is to approximate the sphere with a mesh, as in the quadtree-based hierarchical triangular mesh of Szalay et al. [42]. The application of their method here is limited, however, because the technique was designed primarily for point location queries, does not scale clearly to higher dimensions, and assumes all vectors have equal magnitude.

2.6 Limitations

It should be noted that computational geometry techniques in general and the simplicial partitioning method of Matoušek in particular usually assume that the dimension is less than, say, thirty. So, one should be careful with using our indexing algorithm on vectors of more than thirty components.

Another concern with the techniques described in this chapter is polarity. With regards to the general index, the duality transform involves a division by x_d, so issues of polarity arise when x_d approaches zero. Similarly, the stereographic projection of a point near the pole at $(0, \ldots, 0, 1)$ approaches infinity. For both these scenarios a conceivable approach is to split the index into two. For example, the stereographic projection of points in the positive hemisphere could be done with respect to the negative pole and the stereographic

projection of points in the negative hemisphere could be done with respect to the positive pole, guaranteeing that every image lies within the image of the circumference of the unit sphere.

2.7 Final Remarks

In this chapter we have introduced the novel threshold projection queries (TPQs) and illustrated their advantage over top-k LOQs. Moreover, we focused on designing the first indices to support efficiently retrieving the response to a projection query, those vectors in a dataset with a sufficiently large projection onto an arbitrary query. To enable the techniques in this chapter, we introduced the concept of a *cap*, the component of the unit $(d\text{-}1)$-sphere bounded by a hyperplane orthogonal to a tuple vector and at a distance from the origin proportionate to the vector's magnitude. Conceptually speaking, the cap of a vector is a representation of the set of exactly those queries for which the vector is part of the solution set.

Using this insight permits us to create efficient, novel indices. In general dimension, we employed a duality transform and proven-optimal data structures in previous literature to arrive at an index which requires only $\mathcal{O}(n^{1-1/d+\epsilon} + ts/b)$ blocks of I/O to respond to an arbitrary query.

Next, we recognised that in some scenarios, thresholds are restricted to a small, finite domain. This observation permitted fixing the threshold τ. By fixing τ, we could produce yet more efficiency by spatially indexing the images of the caps under stereographic projection. In two dimensions, we utilised the optimal and well-known interval tree, which guarantees worst-case logarithmic I/O query cost (plus the cost of outputting the result). In three dimensions, we used the asymptotically optimal priority r-tree, which resulted in an index with worst case $\mathcal{O}(\sqrt{ns/b} + ts/b)$ I/O query cost.

Towards our over-achieving objectives in this dissertation, this chapter introduced a new means of formulating preference queries that is better suited to some domains in which the optimal result set cardinality is unclear. In the next chapter, we focus on techniques for mRTOP queries.

Chapter 3

Monochromatic Reverse Top-k Queries and Top-k Rank Contours

A monochromatic reverse top-k (mRTOP) query asks for, given a (possibly new) tuple q and a dataset $\mathcal{D}$, all possible LOQs on $\mathcal{D} \cup \{q\}$ for which q is in the result. We study the novel but practical scenario of designing a layer-based index to respond to two dimensional queries in worst-case-logarithmic time. Our technique is based on identifying a critical *k-polygon* in a dual space representation of the dataset and transforming the query into an intersection test. We implement the two state-of-the-art monochromatic reverse top-k algorithms and perform an extensive experimental evaluation against our technique, demonstrating superior performance in terms of both query cost and memory footprint.

A corollary to the efforts in this chapter to answer mRTOP queries, is the introduction of a new summarization operator, the top-k rank contour. This contour is the identified k-polygon and it exactly encodes the k-ranked tuple for any possible LOQ. While this is certainly significant to the primary objective in this chapter, it is also quite fascinating in its own right, as it represents the minimum possible representation of all information necessary to find the exactly k-ranked tuple without any prior knowledge of the query.

Summary of Chapter 3 Contributions

This chapter makes several significant advances on the state of the art for reverse top-k queries:

- We introduce the first query-agnostic approach. This allows computation to be reused or even done offline in advance. It also implies that the performance is consistent

Symbol	Definition
$\mathcal{D}$	The input, a set of tuples/vectors/points
n	$\lvert\mathcal{D}\rvert$, the number of tuples in $\mathcal{D}$
d	The dimension of the problem
q	The query tuple, of which we ask the effect of adding to $\mathcal{D}$
l_q	The dual space transformation of the query tuple
k	A threshold for the rank of an interesting point of $\mathcal{D}$
$\mathbf{w}$	A vector of attribute weights, presumably supplied by a user
w_i	The i'th coordinate of $\mathbf{w}$
$\mathcal{L}$	A set of lines, obtained by transforming points in $\mathcal{D}$
O	The origin; the point at $(0, \ldots, 0)$
$\mathtt{P}_k$	The critical k-polygon
$\mathtt{H}$	The convex hull of $\mathtt{P}_k$

Table 3.1: Table of repeatedly used notation for Chapter 3

regardless of the query point. Previous work is sensitive both to the dataset *and* the choice of query point;

- We introduce new geometric techniques for the problem that are themselves of high interest: our novel depth contours and k-polygons provide new tools that are useful to researchers investigating other related top-k and reverse top-k problems;

- And we demonstrate consistently better empirical performance than other algorithms, often by an order of magnitude, indicating practical utility for the ideas we present.

3.1 Preliminaries

In this chapter, we study monochromatic reverse top-k (mRTOP) queries and introduce top-k rank contours. As argued in Section 1.2, these queries are an interesting way of evaluating the global relevance of a particular tuple. In this section we formally introduce the problem under study and the concepts upon which the chapter depends. Note, however, that we assume familiarity with a few concepts defined in Section 2.1, such as *projection* and *duality*.

Throughout this entire chapter, we assume queries are executed on a two-dimensional, numeric relation $\mathcal{D}$ which is a set of tuples $(v_1 \in \mathbb{R}, v_2 \in \mathbb{R})$. Tuples can also alternatively be viewed as points (v_1, v_2) in the Euclidean plane or as two-dimensional vectors $\vec{v} = \langle v_1, v_2 \rangle$. We assume $\lvert\mathcal{D}\rvert$ is "large", and that k is a small constant, $k \in \mathbb{Z}^+ \ll \lvert\mathcal{D}\rvert$.

To begin, a *traditional, linear* $\text{top-}k$ *query* (or LOQ, as we have referred to it before) is a pair of weights w_1, w_2. The *response* is the set of k tuples in $\mathcal{D}$, which, when interpreted as vectors, have the largest dot product with $\langle w_1, w_2 \rangle$. That is to say:

Definition 3.1.1. *The response to a* traditional, linear $\text{top-}k$ query, $\vec{w} = \langle w_1, w_2 \rangle$, *is the set:*

$$TOP(\vec{w}) = \{\vec{v} \in \mathcal{D} : |\{\vec{u} \in \mathcal{D} : \vec{u} \cdot \vec{w} > \vec{v} \cdot \vec{w}\}| < k\}.$$

For example, considering the sample dataset given back in Figure 1.1, the result of a top-2 query for weights $w_1 = .5$, $w_2 = .5$ is the set $\{p_1, q\}$. These are the two tuples with highest weighted averages according to $\vec{w}$, scoring 0.667 and 0.563, respectively.

It is worth noting that in some cases more than k tuples may satisfy the conditions of Definition 3.1.1 if they produce the same-sized dot product with the query vector (that is to say, if they "tie"). For a $\text{top-}k$ query, these ties are broken *arbitrarily*. Of those tuples tied with the smallest dot product, exactly enough are selected to ensure the output has exactly k tuples.

The *monochromatic reverse* $\text{top-}k$ *query*, introduced by Vlachou et al. [46], which we refer to simply as a *reverse* $\text{top-}k$ *query* for much of this chapter, is a query tuple $q = (q_1, q_2)$ not necessarily in $\mathcal{D}$. The response is the set of traditional, linear $\text{top-}k$ queries on $\mathcal{D} \cup \{q\}$ for which q is in the result set. Formally:

Definition 3.1.2. *The response to a* reverse $\text{top-}k$ query, $q = (q_1, q_2)$, *is the set of angles*

$$\begin{aligned} RTOP(q) \;=\; & \{\theta \in [0, \pi/2] : \\ & |\{v \in \mathcal{D} : v_1 + v_2 \tan\theta > q_1 + q_2 \tan\theta\}| \\ & < k\}. \end{aligned}$$

For example, given the query depicted in Table 1.1, $q = (0.725, 0.400)$ the reverse top-2 query response are the shaded angles in the set $[0, 0.184\pi] \bigcup [0.424\pi, 0.444\pi]$. Any set of weights w_1, w_2 that corresponds to a vector with angle from the positive x-axis in those ranges will produce q as a member of the top-2 query for $\vec{w}$ on $\mathcal{D} \cup \{q\}$.

Note that in the case of Definition 3.1.2, sets of weights that lead to the query tuple "tying" are included in the result.

Additionally to these problem definitions, we define here a number of concepts with which in the subsequent sections we assume the reader is familiar. Specifically, we define

here the *nullspace* of a vector, an *arrangement of lines*, and our novel concepts of *top-k rank* and *top-k rank contours*.

Definition 3.1.3. *The* nullspace *of a vector $\vec{v} = \langle v_1, v_2 \rangle$ is the set of vectors orthogonal to $\vec{v}$: $\{\vec{u} : \vec{u} \cdot \vec{v} = 0\}$. In two dimensions, this is exactly the line $y = -\frac{v_1}{v_2}x$. The* translated nullspace *of $\vec{v}$, given a positive real τ, is the set of vectors $\{\vec{u} : \vec{u} \cdot \vec{v} = \tau\}$, or the line $y = \frac{\tau}{v_2} - \frac{v_1}{v_2}x$.*

Vector nullspaces are important in this chapter, because we transform every point into its translated nullspace (a line), and then process the resulting arrangement of lines.

Definition 3.1.4. *An* arrangement *of a set of lines $\mathcal{L}$, denoted $\mathcal{A}_\mathcal{L}$, is a partitioning of $\mathbb{R}^2$ into cells, edges, and vertices. Each* cell *is a connected component of $\mathbb{R}^2 \setminus \mathcal{L}$. Each vertex is an intersection point of some two lines $l_1, l_2 \in \mathcal{L}$. An edge is a line segment between two vertices of $\mathcal{A}$.*

Definition 3.1.5. *The* top-k rank *of a point p within an arrangement $\mathcal{A}$, is the number of edges of $\mathcal{A}$ between p and the origin. That is to say, the depth of p is the number of intersections between edges of $\mathcal{A}$ and (O, p). Similarly, the* top-k rank *of a cell of $\mathcal{A}$ is the top-k rank of every point within that cell.*

We will show later that Definition 3.1.5 implies that edges also have a unique depth, because depth can only change at vertices. So, we define contours in terms of the depth of edges:

Definition 3.1.6 (Top-k Rank Contour)**.** *A* top-k rank contour *is the set of edges in an arrangement $\mathcal{A}_\mathcal{L}$ that have top-k rank exactly k. We also refer to a* top-k rank contour *as the k-*polygon *of $\mathcal{L}$, because, as we show later, the contour is a closed, star-shaped polygon.*

3.2 An Arrangement View

The theme of this chapter is to answer monochromatic reverse top-k (mRTOP) queries with *logarithmic cost* by means of a data structure featuring a largely sequential data layout and inspired by geometric analysis of the problem. In this section, we conduct that analysis and create the theoretical foundations for our correctness proof of our access methods in Section 3.3.

The approach taken in Vlachou et al. [46] is to exploit the dominance relationship among points in $\mathcal{D}$. The approach taken in Wang et al. [50] is to compare all points in $\mathcal{D}$ to the query point q in the dual space. We take a very different approach. We transform the dataset into an arrangement of lines and demonstrate that embedded in the arrangement is a critical polygon $\mathcal{P}_k$ which partitions $\mathbb{R}^2$ into points to include among and exclude from a mRTOP query result. We show, too, that by applying the same transformation to the query to produce a line l_q, $mRTOP(q)$ is given precisely by the intersection of l_q with the interior of $\mathcal{P}_k$.

An equally important contribution of this section is that we derive properties of $\mathcal{P}_k$ that are critical for proving later the asymptotic performance of our access method.

This section is thus divided into three subsections: the first describes the transformation of $\mathcal{D}$ into a set of $|\mathcal{D}|$ contours (Section 3.2.1); the second derives important properties of $\mathcal{P}_k$ (Section 3.2.2); and the third establishes the equivalence of the intersection test to the original mRTOP problem (Section 3.2.3).

3.2.1 $\mathcal{A}_\mathcal{L}$ and Top-k Rank Depth Contours

In this section we describe what is a top-k rank contour and how it is constructed from a relation, $\mathcal{D}$. We illustrate how to construct the arrangement from $\mathcal{D}$ and how to interpret the arrangement as a set of contours. First, in order to reason about $\mathcal{D}$ in terms of an arrangement, we need to represent each tuple as a line such that the relative positions of the lines with respect to a ray from the origin reflects their top-k ranking. This is precisely the property that is proferred by the translated nullspaces of each tuple, for any arbitrary real τ.

So, we convert the set of tuples (or, alternatively, vectors) $\mathcal{D}$ into a set of lines by transforming each tuple $v = (v_1, v_2)$ to the line $\overline{v} : y = \frac{\tau}{v_2} - \frac{v_1}{v_2}x$. For a ray r in any direction, we can show that:

Lemma 3.2.1. *If the depth of a point v is less than the depth of a point u in the direction of a ray r, then the rank of v for a traditional, linear top-k query $\vec{r}$ is better than that of u.*

Proof If the translated nullspace of $\vec{v}$ is closer to the origin than of $\vec{u}$ in the direction of r, then $\vec{v} \cdot \vec{r} = \tau = \vec{u} \cdot c\vec{r}$ for some $c > 1$. Therefore, $\vec{v} \cdot \vec{r} > \vec{u} \cdot \vec{r}$. □

In fact, we can make a stronger claim: the depth of a point p is precisely its top-k rank for a query in the direction of p if p happens to correspond to a point on an edge of the arrangement.

Corollary 3.2.2. $depth(p) = \text{rank}(\vec{p})$ *for* $TOP(\vec{p})$.

Proof Let $depth(p)$ be d. Then from the definition of top-k rank there are d other base-planes that will be sooner encountered by a ray emanating from O in the direction of p. From Lemma 3.2.1, we know that each of these has a better rank than p, so the rank of p is at best d. Also, from Lemma 3.2.1 we can conclude that p has a better rank than all those with translated nullspaces farther from the origin than that of $\vec{p}$, so the rank of p is not greater than d, either. □

The k'th contour of an arrangement is the set of all edges at the same depth. We wish to show that, in fact, the edges form a connected ring around the origin, thus forming a polygon. In order for this to be true, we need to show that in any direction there is exactly one point on the contour, and that the points are all adjacent to each other. This is the objective of the following three lemmata.

Firstly, to demonstrate connectedness, it is important that top-k rank is a monotone measure:

Lemma 3.2.3. *Top-k rank depth increases monotonically with Euclidean distance from O in any arbitrary direction.*

Proof Consider two points p, q such that p lies on the line segment $[O, q]$. Every line in the arrangement that crosses $[O, p]$ also crosses $[O, q]$, so $depth(q) \geq depth(p)$. □

Secondly, we need to show that a cell of depth i is unique in a given direction:

Lemma 3.2.4. *There is exactly one cell of depth i in any given direction from O, for reasonably small i.*

Proof First, we show that there is at most one cell of depth i. This follows from the definition of top-k rank. Assume for the sake of contradiction that there are two disjoint cells, A and B, with depth i in the same direction. Without loss of generality, assume that A is nearer to O than B. Take some point $a \in A$. Then, from the definition of top-k rank, we know that there are exactly i lines crossing the line segment $[O, a]$. Now consider some point $b \in B$. Because A is nearer than B to O, clearly every line between a and O also crosses the line segment $[O, b]$. So, too, must the upper boundary of A, since A and B are distinct. But then there are at least $i + 1$ lines crossing $[O, b]$, which contradicts that B is at depth i.

The assumption that i is reasonably small is to guarantee that there are sufficiently many tuples in $\mathcal{D}$ that there are at least i tuples to return for a traditional $\text{top-}k$ query. This

is enough to imply that there is an i-contour in every possible direction, so there must be at least one cell in our given direction at depth i, as well. □

Thirdly, we can now show that, in fact, all cells of depth i are connected and can thus form a contour:

Corollary 3.2.5. *All cells at the same top-k rank ($\leq k_{\max}$) are connected.*

Proof This follows from Lemma 3.2.4, which implies that there are no discontinuities in the contour in any given direction. Observe, too, that for any cell there must be an adjacent cell with the same depth at every corner. The corners correspond to directions in which the incident translated nullspaces reverse order. So, since the top translated nullspace becomes a bottom translated nullspace and vice versa, the depth does not change.[1] □

This is enough to establish that the k'th contour of the arrangement is precisely a star-shaped polygon:

Theorem 4. *A contour is a star-shaped polygon.*

Proof First, we know that the contour is connected and exists in every direction from O. Also, every point inside the polygon is visible from O, for if there were some point p that were not visible, then an edge of the boundary would cross $[O, p]$. However, this would imply that there are two cells at the same depth in the direction of p from O, contradicting Lemma 3.2.4. □

Theorem 4 establishes that we can represent $\mathcal{D}$ as a set of polygons with a unique depth i, each of which itself encodes the i'th ranked tuple for any possible traditional, linear top-k query. If there is only one value k of interest, then the entire dataset can be represented just by one polygon. In this next subsection, we show properties of the k-polygon, including bounds on its size, and in the following subsection describe how to use it in order to address the main question of this chapter, mRTOP queries.

3.2.2 Properties of $\mathcal{P}_k$

In order to be able to use $\mathcal{P}_k$ as a data structure, we have to evaluate properties of the polygon in order to evaluate asymptotic performance. As we will detail in the next section, our data structure will be a representation of $\mathcal{P}_k$, so the number of edges and vertices in the polygon influences our access time.

[1] Strictly speaking, the vertex/corner itself is a discontinuity, as there is no point in that direction with exactly the right number of crossing line segments, but this is infinitesimal in size and we ignore the issue because we return open intervals anyway.

Also, to improve performance, our data structure includes a convex approximation of $\mathcal{P}_k$ (specifically the convex hull), and understanding the implications of this approximation is also important.

Thirdly, we approximate the dataset $\mathcal{D}$ by $\mathcal{S}_k$, so understanding the implications of this approximation is clearly important, as well.

Gathering this understanding is the intent of these next three lemma. Specifically, they answer these three questions in order:

Proposition 3.2.6. *An arrangement of m lines can produce contours at top-k rank i with no more than $\mathcal{O}(m)$ edges.*

Lemma 3.2.7. *A concave region between vertices of the convex hull of the k'th contour's upper boundary can have at most $2k-1$ vertices.*

Proof Notice that vertices of the convex hull of the contour's upper boundary are themselves at depth $k-1$. Consider two such vertices, v_i, v_j, delimiting a concave region. Any line that passes neither under v_i nor under v_j and is orthogonal to some non-zero vector from O cannot pass through the concave region's face, so the face is defined by at most $2k$ lines. This is, in fact, an arrangement, so Proposition 3.2.6 implies the bound on the number of cells in that arrangement that could possibly be at depth k and thus contribute a vertex to the concave region's boundary. □

3.2.3 A Transformed mRTOP Query

In the previous subsections we have demonstrated that a star-shaped polygon (the k-polygon) can encode the k'th best ranked tuple for all query directions. In this section, we demonstrate how to use the k-polygon for mRTOP queries.

First, recall that the arrangement of lines was produced by transforming each tuple in $\mathcal{D}$ to its translated nullspace, given some fixed but arbitrary τ. Here, we prove that applying the same transformation to a query q to produce a line l_q and intersecting l_q with the interior of $\mathcal{P}_k$ yields the directions in which q is among the result set of traditional, linear $\text{top-}k$ queries:

Theorem 5 (Dual space equivalence of a mRTOP query)**.** *The response to a mRTOP query, given query vector $\vec{q} = \langle q_1, q_2 \rangle$, is the component of $\overline{q} : y = \frac{\tau}{q_2} - \frac{q_1}{q_2}x$ which intersects the interior of $\mathcal{P}_k$.*

Proof Recall from Theorem 3.2.2 that the k'th contour corresponds exactly to the vectors of rank k and also from Lemma 3.2.3 that the contours increase in rank monotonically. Therefore, if we constructed a new arrangement which also contained $\overline{q}$, the components of $\overline{q}$ which lay outside the k'th contour would be directions in which the rank of q is greater than k. The inverse of this is the solution to the mRTOP query. □

Consequently, it suffices to develop algorithms for solving the problem of identifying the segments of $\overline{q}$ which lie inside the k'th top-k rank contour in order to solve the mRTOP problem.

A final note regarding the properties of $\mathcal{P}_k$ is that:

Proposition 3.2.8. *The result in two dimensions of a mRTOP query consists of at most two continuous intervals.*

3.3 Efficiently Answering mRTOP Queries

Having established the theoretical foundations in the previous section, we present here our index structure and access method. A key insight that we derived earlier is that the mRTOP response to q is the intersection of l_q with the interior of $\mathcal{P}_k$. Fittingly, then, our index structure is a representation of $\mathcal{P}_k$ and our access method is an efficient means of retrieving from the index the intersection points of l_q with $\mathcal{P}_k$. First we give a high-level overview of our algorithms and data structure and then present the precise details in the upcoming subsections.

Not just any representation of $\mathcal{P}_k$ will suffice: it has to facilitate the efficiency of the access method. We accomplish this by creating a binary search procedure to identify the intersections of l_q with the convex hull of $\mathcal{P}_k$. This leads to an efficient access method because we established Lemma 3.2.7. We have developed a very sequential data structure consisting of one ordered list of the vertices of the convex hull of $\mathcal{P}_k$ and one ordered list of ordered lists of $\mathcal{P}_k$ vertices not on the convex hull. We describe the index structure in Section 3.3.1.

From an algorithmic perspective, there are two main considerations. Of foremost importance is how to efficiently query the index structure, given l_q (Section 3.3.3). The second consideration is how to efficiently construct (Section 3.3.2) it. We begin by addressing the first.

The idea is to exploit properties of the problem. Our binary search to discover the intersection points of l_q with a convex polygon is of logarithmic cost. Furthermore, given the intersection points of l_q with the convex hull of $\mathcal{P}_k$, we can find the exact intersection of l_q with $\mathcal{P}_k$ by comparing it with every edge "shaved off" by that convex hull edge. By Lemma 3.2.7, we know there are most $\mathcal{O}(k)$ such edges. Because of our sequential layout, a direct comparison to each of these $\mathcal{O}(k)$ edges is affordable.

Our construction algorithm is a plane sweep algorithm. We sweep radially from the positive x-axis to the positive y-axis, maintaining a list of all the lines in sorted order with respect to their intersection points on the sweep line. At any given moment during the plane sweep, the k'th line in the list is the edge of the k-polygon. So, identifying the k-polygon is equivalent to identifying all the points at which the k'th line in that list changes. These points are the vertices of the k-polygon. Maintaining the convex hull of the polygon is fairly straight-forward if one maintains convexity as an invariant throughout the sweep.

The expense of this *construction* algorithm is dominated by two factors: processing cells of the arrangement[2] and initially sorting all the lines with respect to their intersection points with the x-axis. Regarding the latter, we can improve upon the cost by recognising that any tuple on the k-contour of $\mathcal{D}$ must intersect the k-contour of any subset of $\mathcal{D}$. So, at the cost of one extra sequential scan, we prune $\mathcal{D}$ with perfect recall (i.e., ensure every true positive is in the approximation) and then construct $\mathcal{P}_k$ from that approximation, rather than from all of $\mathcal{D}$.

The approximation method exploits the work we have already done in this chapter. We build our index structure on $2k$ selected tuples from $\mathcal{D}$ and then include in our approximation any tuples which have non-null mRTOP query responses on that small index structure. The cost of this approximation is one sequential scan at $\mathcal{O}(n\ log\ k)$.

Together, these algorithms and this data structure gives Theorem 6, in which $n = |\mathcal{D}|$:

Theorem 6 (Asymptotics of $2d$ mRTOP)**.** *The two dimensional mRTOP problem can be solved using $\mathcal{O}(\log n + k)$ query time with an index that requires $\mathcal{O}(n)$ disk space.*

Since k is typically a small constant, the above theorem implies that the query cost is $\mathcal{O}(\log n)$.

[2]Note that while in a general arrangement of n lines, there are at most $n + n(n-1)/2$ cells, one for each intersection point and an additional one for each line, we are only interested in those in the positive quadrant.

3.3.1 The k-Polygon Index Structure

Facilitating logarithmic query time of the index largely depends on how the data is represented. Our idea is to exploit Lemma 3.2.7 in our representation. Let $\mathcal{H}$ denote the set of vertices of the convex hull of a k-polygon, $\mathcal{P}_k$. We maintain two arrays, which we collectively refer to as the *dual-array representation* of $\mathcal{P}_k$. The first, which we call the *convex hull array*, contains the $|\mathcal{H}|$ vertices of $\mathcal{H}$, ordered anti-clockwise from the positive x-axis. The second array, which we call the *concavity array*, is of size $|\mathcal{H}| - 1$. The i'th entry contains a sequential list of the up to $2k - 1$ vertices of the k-polygon between the i'th and $(i + 1)$'st vertices of $\mathcal{H}$.

3.3.2 Construction of the k-Polygon

Although Section 3.2.2 suggests how to determine the k-polygon of $\mathcal{D}$ by first constructing an arrangement of lines and then extracting from it all the edges at a top-k rank of k, here we describe a much more efficient algorithm. First we approximate the dataset with perfect recall. We then sort the remaining lines based on their x-intercept. Finally, we conduct a radial plane sweep algorithm to build the polygon index.

Dataset Approximation

The important consideration in our dataset approximation is that perfect recall is critical. Otherwise, we may miss a line that forms part of the k-contour. We exploit the insight that the k best lines with respect to each axis form a contour relatively close to the real contour, and that if a tuple is in the k-contour, it clearly must be in the k-contour of any subset of the data. Thus, the approximation algorithm proceeds by quickly determining the $\leq 2k$ lines as above, constructing a contour from them, and determining which lines in $\mathcal{D}$ have non-null mRTOP query answers on the approximate contour. See Algorithm 3.

Radial Plane Sweep

We construct a contour from a set of lines using a radial plane sweep. The idea is to traverse the set of intersection points in angular order, maintaining a sorted list of the lines. In this way, we build the contour incrementally from the positive x-axis towards the positive y-axis. Traversing in this order also allows us to maintain convexity of the contour as we go. Like most plane sweeps, a primary advantage is that we need only look at intersection points between two lines after they become neighbours. If this does not occur between the

Algorithm 3 Approximating $\mathcal{D}$

Input: $\mathcal{D}$; k
Output: $\mathcal{S} \subseteq \mathcal{D}$, the tuples that form the k-contour of $\mathcal{D}$, plus potentially some false-positives
Initialise $\mathcal{S}$, an empty set of tuples
Let $\mathcal{X}$ denote the k tuples in $\mathcal{D}$ with the highest values for attribute x
Let $\mathcal{Y}$ denote the k tuples in $\mathcal{D}$ with the highest values for attribute y
Construct $\mathcal{P}_{\mathcal{X}\cup\mathcal{Y}}$, the k-polygon index on the set $\mathcal{X} \cup \mathcal{Y}$ using Algorithm 4.
for all $p \in \mathcal{D}$ **do**
    if l_p intersects the interior of $\mathcal{P}_{\mathcal{X}\cup\mathcal{Y}}$ or $p \in \mathcal{X} \cup \mathcal{Y}$ **then**
        Add p to $\mathcal{S}$
    end if
end for
Free $\mathcal{X}$ and $\mathcal{Y}$.
RETURN $\mathcal{S}$.

sweep line and the positive y-axis, then we need not consider the intersection point at all. Algorithm 4 offers the details of the sweep algorithm.

3.3.3 Querying the k-Polygon Index

Here we present how to query our k-polygon index to determine the segments of a line l_q that are strictly contained within the interior of the k-polygon, $\mathcal{P}_k$. The algorithm (Algorithm 5) is a binary search on the convex hull of the polygon, proceeded by a sequential scan of $\mathcal{O}(k)$ edges of $\mathcal{P}_k$. The recursion is based on the slope of l_q compared to the convex hull of $\mathcal{P}_k$ at the recursion point.

3.3.4 Asymptotic Performance

Earlier we stated the asymptotic performance of our algorithms. Here, now, we have the tools to prove that theorem. The basic idea is that a line can only intersect a convex shape in two locations and for each of those intersection points, the cost of a face traversal is bounded.

Proof of Theorem 6 First, note that a line can only intersect the boundary of a convex polygon in at most two points, so the binary search tree traversal need follow at most two paths. Recall from Lemma 3.2.6 that each contour contains at most n cells, and thus the convex hull contains at most $n - 1$ edges. From Lemma 3.3.1, the binary search requires $\mathcal{O}(\log n)$. For each of the two intersection points found, we traverse the corresponding

Algorithm 4 Building $\mathcal{P}_k$

Input: $\mathcal{L}$, an array of lines sorted by ascending x-intercept; k
Output: A dual-array representation of $\mathcal{P}_k$
Initialise an empty array $\mathcal{H}$ for convex hull vertices
Initialise an empty array of lists $\mathcal{C}$ for concavities
Initialise $\mathcal{I}$ as a priority queue containing the $|\mathcal{L}| - 1$ intersections of neighbouring lines in $\mathcal{L}$, sorted by angle from the positive x-axis, discarding those < 0.
while $\mathcal{I}$ is not empty **do**
    Pop next intersection $i \in \mathcal{I}$
    Let l_{left} and l_{right} be the lines intersecting at i.
    if $l_{left} = \mathcal{L}_{k-1}$ or $l_{right} = \mathcal{L}_{k-1}$ **then**
        Add i to $\mathcal{H}$
        if $\exists h \in \mathcal{H} : \text{slope}([h, i]) < \text{slope}([h, h + 1])$ **then**
            Add to $\mathcal{C}_h$ all vertices between h and i.
            Remove all vertices between h and i from $\mathcal{H}$ and from $\mathcal{C}_j, \forall j \neq h$.
        end if
    end if
    Swap l_{left} and l_{right} in $\mathcal{L}$
    Add to $\mathcal{I}$ the intersection of l_{left} with its new neighbouring line and the intersection of l_{right} with its new neighbouring line, provided they are at angles greater than that of i and in the positive quadrant
end while
Free $\mathcal{I}$.
RETURN $\mathcal{H}$ and $\mathcal{C}$.

face sequentially. From Lemma 3.2.7, each of these faces contains $\mathcal{O}(k)$ edges and we know that finding the intersection (or, equivalently, ascertaining the non-intersection) of two two-dimensional line segments requires constant time.

Since the search is run independently of and its cost dominates the cost of the face traversals, and since k is a small constant (i.e., $\mathcal{O}(1)$), the entire query procedure is $\mathcal{O}(\log n)$.

Regarding the space requirements, Lemma 3.2.6 implies that polygon itself can contain at most $\mathcal{O}(n)$ vertices. Because each vertex could appear at most twice in the data structure (one on the convex hull and once in a single concavity), and because the data structure is, simply, the vertices of the k-polygon, the disk space required by the data structure is $\mathcal{O}(n)$. □

Lemma 3.3.1. *The intersection of the query line with the convex hull can be determined in* $\mathcal{O}(\log n)$ *time.*

Proof The intersection algorithm proceeds by binary search. First, find the middle vertex

Algorithm 5 Querying a dual-array k-polygon, $\mathcal{P}_k$

Input: Dual-array representation of $\mathcal{P}_k$, line l_q, start/end indexes.
Output: Intersection points of l_q with $\mathcal{P}_k$
if $end - start = 2$ **then**
    Traverse the $\mathcal{O}(k)$ list in the concavity array at position $start$, returning any intersections with l_q.
    RETURN.
end if
Compute midpoint vertex of $\mathcal{H}$ at $\frac{end-start}{2} + start$.
if l_q passes above midpoint **then**
    if slope of l_q is less than slope of [midpoint-1, midpoint] **then**
        Recurse on lower half with end=midpoint
    else if slope of l_q is greater than slope of [midpoint, midpoint+1] **then**
        Recurse on upper half with start=midpoint
    end if
else
    if l_q passes above vertex at position start **then**
        Recurse on lower half with end=midpoint
    end if
    if l_q passes above vertex at position end **then**
        Recurse on upper half with start=midpoint
    end if
end if

$v_{n/2}$ and determine whether the query line passes above or below it. If above then recurse left if the query line has shallower slope than edge $(v_{n/2}, v_{n/2+1})$. Recurse right if the query line has steeper slope than edge $(v_{n/2-1}, v_{n/2})$. Because edge $(v_{n/2}, v_{n/2+1})$ is shallower than edge $(v_{n/2-1}, v_{n/2})$, at most one recursion direction can be followed.

If, instead, the query line passes below $v_{n/2}$, then it is inside the contour (if in the correct quadrant at all). To find the intersection points, recurse left if the query line passes above v_{n-1}. Recurse right if the query line passes above v_0. It is possible that both conditions are true, but this can only occur once, because the truth of the condition implies an intersection point and a straight line has at most two intersection points with a convex polygon. Therefore, the binary search follows at most two distinct paths. □

3.4 Empirical Investigation

In this section we augment the theoretical guarantees of the previous sections with an empirical investigation. We compare the use of our index to that of repeatedly executing the

state-of-the-art algorithms of Vlachou et al. [46] and of Wang et al. [50]. In Section 3.4.1 we describe the methodology of the experiments; in Section 3.4.2 we give the results of each experiment; and in Section 3.4.3 we draw conclusions from the results.

3.4.1 Setup

Our experiments, in total, consist of thousands of queries to address six key questions across eight datasets, three of which are new and that we collected specifically for these experiments.

Questions

We evaluate the performance of our proposed algorithm along two directions: how it performs against other known algorithms and how it scales as the data grows. This leads to six specific questions that we investigate:

Q1. How long does it take to answer a query (wall time)?

Q2. How many IOs are needed to answer a query?

Q3. How much memory is consumed in answering a query?

Q4. How long does it take to construct the index?

Q5. How large is the index data structure, once created?

Q6. How elegantly does the query cost degrade as the size of the dataset is scaled up?

Datasets

For the experiments, we make use of eight datasets, which are summarized in Table 4.3. Of these, five are distinct projections of one larger, publicly available dataset and the other three we created from online sources.

N1-5: NBA statistics: Datasets N1-5 are five separate projections of the regular season statistics from *databasebasketball.com*. We reserve the most recent season of the dataset, 2009, as a set of 578 query points and use the other seasons, 1946-2008, as the dataset of 21383 tuples. Each mRTOP query asks which blends of the given two skills, if any, was this particular player's performance this season ranked among the top-k of all-time. This

$\mathcal{D}$	$\|\|\mathcal{D}\|\|$	x-Attribute	y-Attribute	Corr.	Corr.1%
N1	$21,383$	Points	Field goals made	0.999	0.891
N2	$21,383$	Defensive rebounds	Blocks	0.749	0.157
N3	$21,383$	Personal fouls	Free throw attempts	0.749	-0.176
N4	$21,383$	Defensive rebounds	Assists	0.328	0.082
N5	$21,383$	Blocks	Three pointers made	0.025	-0.031
M	$5,000$	Replies	Views	0.858	0.590
A	400	Price	Rating	-0.658	N/A
W	363	Mean	Standard deviation	0.687	N/A

Table 3.2: *Datasets under study.* Each of the eight datasets ($\mathcal{D}$) under study: their size, normalized attributes and Pearson correlation coefficient. For the larger datasets, the Pearson correlation coefficient for the top 1% (relative to a sort on the y-attribute), is given, because these "top" records most heavily influence the size of the output and, hence, the expected performance of any algorithm.

contrasts to traditional analysis of sports statistics, which fails to capture players who excel in combinations of skills yet do not make the top rankings for each skill alone.

M. Message board threads: Dataset M is a set of 5495 message board threads, each described by a two-tuple $(\text{replies}, \text{views})$. We collected the data from *TnFNorth*,[3] a now-dormant Canadian track and field message board that was fairly busy during its lifespan from 2004-2010 before migrating to its current new URL. These two numeric attributes are particularly interesting because they capture the two considerations especially relevant to advertisers: the degree of user contribution and the traffic. Identifying top-ranked threads with respect to these parametres can help to drive revenue by inspiring new threads that are similar in nature.

We reserve the 495 most recently updated threads as query points to execute against the 5000 older points. A reverse $\text{top-}k$ query, then, asks for what blend of contribution and traffic is this thread interesting in a historical context. One can imagine an automation of these queries whenever a thread is viewed or answered which notifies the moderator when threads are generating a significant amount of contribution and/or traffic and captures interesting cases that the two attributes would independently miss.

A. All-inclusive deals: Dataset A is a set of 400 seven-day, all-inclusive vacation package deals departing from Vancouver International Airport (YVR), each described by a two-tuple $(\text{price}, \text{rating})$. We collected the data from a popular, local aggregation website, *yvrdeals.com*,[4] which retrieves the prices (per person, including taxes) from the package providers and the (average reviewer) ratings from *TripAdvisor.com*. The aggregation site features a built-in ranking mechanism and displays 10 deals per page.

We normalize both attributes by dividing the price by 2500 (slightly larger than the maximum collected) and the rating by 5.0 (since the ratings are on a 1-5 scale). Five days later we returned to the site to collect some query points by issuing the same search as before and collecting 25 deals that were new. As such, a reverse top-$10i$ query is equivalent to asking whether this new deal would appear on the i'th page for any user's search results, if the user could specify his/her own weights.

As an important aside, we do not know the underlying ranking mechanism on the aggregation site. However, we computed a ranking of the deals with weights set to $< .5, .5 >$ for the two normalized attributes that we collected. We then performed a linear regression

[3] `http://tnfnorth.proboards.com/index.cgi?board=general`, data retrieved 16-Feb-2013.

[4] `http://www.yvrdeals.com/all-inclusive-vacations-from-Vancouver/`, data retrieved 14-Feb-2013.

on the website ranking and our ranking and observed a Pearson Correlation of 0.998. Our conclusion is that whatever ranking mechanism is in place on this popular site, it can be very closely approximated with a simple linear aggregation of these two primary attributes. This corroborates our opinion that, after the application of boundedly-many predictable filters, vacation deals are a trade-off of just price versus quality.

W. Website responsiveness distributions: Dataset W is a set of 363 normal distributions, each described by a two-tuple (μ, σ). We collected the top 500 website URLs from the global Internet traffic ranking at *Alexa.com*.[5] For each, we measured the response time by issuing *pings* 100 separate times (to invoke the Central Limit Theorem) spread out over 26 hours from a consistent Victoria, Canada-based IP address. We then computed the mean and standard deviation for each URL and normalized the two attributes to a [0, 1] range across the data set. Of the original 500 URLs, 12 could not be resolved, and 125 were unresponsive or timed out (on default *ping* settings).

Hence, the dataset describes *responsiveness* and *consistency*, respectively. The attributes have a Pearson correlation coefficient of 0.68. A reverse $\text{top-}k$ query on W asks for which balances of *responsiveness* and *consistency* a query URL ranks among the top k in comparison to the highest traffic URLs on the Internet. For the query set, we compute the same metrics for 50 arbitrarily chosen institutional sites from Canada. One can imagine creating this index for an online web service for ad-hoc querying the relative performance of user's own sites.

Of course, many two-dimensional datasets could be created in a similar fashion from repeated observations of independent phenomena.

Configuration

We implemented and optimised the algorithms of Vlachou et al., of Wang et al., and of the authors (Chester et al.) in C and compiled our implementations with the GNU C compiler 4.4.5 using the *-O6* flag. We ran the experiments on a machine with an Intel Atom processor with two 800MHz cores and 1GB RAM, running Ubuntu. The timings were calculated using the Linux *time* command and include the cost of reading the data (which is done differently by each algorithm), but not of displaying the output (which is roughly the same). The data structure sizes were measured in separate trials by modifying the code to count the number of vertices created. The memory footprints were measured by noting the allocations reported by *Valgrind*. The I/O values were read from the *syscr* value in the

[5] `http://www.alexa.com/topsites`, retrieved 17-Feb-2013.

$\mathcal{D}$	Metric	k=10		k=20		k=30		k=50		k=100	
N1	W	153.6	6.3	293.6	5.1	463.6	6.9	690.8	7.8	1,237.2	6.5
N1	V	62.8	3.8	113.2	4.2	159.6	5.8	333.6	5.7	565.2	6.3
N1	C	4.8	2.5	6.4	2.8	5.6	2.8	5.6	2.1	6.4	2.1
N2	W	47.6	5.5	67.6	3.5	95.6	6.4	230.0	4.3	377.2	7.1
N2	V	66.4	4.3	200.8	5.6	246.0	5.1	342.0	3.9	599.1	9.2
N2	C	6.0	2.8	5.6	3.4	6.4	2.1	4.4	3.5	5.6	2.1
N3	W	48.4	5.1	78.0	2.8	192.8	7.5	341.2	7.1	601.6	6.3
N3	V	157.2	6.0	208.8	7.0	256.8	4.1	447.6	5.4	885.6	9.3
N3	C	5.6	2.8	4.8	4.1	3.6	3.0	4.8	4.5	5.2	2.7
N4	W	127.2	6.2	224.8	4.9	431.6	6.9	676.4	7.9	1,737.6	7.8
N4	V	242.4	6.3	524.4	7.4	834.0	12.8	1,384.4	11.8	2,266.0	21.4
N4	C	5.2	1.9	6.0	2.8	5.6	2.1	6.4	2.8	4.4	2.3
N5	W	321.2	9.6	632.8	7.7	988.8	10.5	1,512.0	11.9	2,931.6	26.4
N5	V	683.2	7.0	1,340.8	9.0	1,725.2	18.2	2,756.8	20.0	5,103.2	127.9
N5	C	6.0	2.1	6.4	3.4	4.0	2.7	5.6	2.8	4.8	3.7
M	W	348.8	6.2	457.6	9.7	574.4	13.4	754.4	5.7	N/A	
M	V	266.0	5.7	371.2	4.5	479.2	6.7	680.8	6.7	N/A	
M	C	2.4	2.8	1.6	2.1	1.2	1.9	2.0	2.1	N/A	
A	W	7.2	3.7	20.8	3.2	21.6	5.1	N/A		N/A	
A	V	40.4	7.2	45.2	8.4	43.2	3.7	N/A		N/A	
A	C	0.7	1.7	0.4	1.3	0.4	1.3	N/A		N/A	
W	W	32.0	3.8	36.4	3.5	43.6	4.8	N/A		N/A	
W	V	6.8	2.7	6.8	3.3	11.2	3.7	N/A		N/A	
W	C	1.6	2.1	2.8	2.7	0.8	1.7	N/A		N/A	

Table 3.3: *Experiment results: query wall time.* The cost, measured in milliseconds, to execute the batch of queries. In each cell the left value is the mean of ten trials and the right value is the standard deviation. "N/A" indicates values of k not run.

/proc/PID/io file.

3.4.2 Results

In this subsection we disclose the raw (aggregated) results of the experiments detailed in Sect. 3.4.1. In the next subsection we will draw inferences from the results and present representative trend figures.

We begin with the simplest question: how fast are the algorithms. In Table 3.3 we give the time to complete each batch of queries for each algorithm on each dataset for values of $k \in \{10, 20, 30, 50, 100\}$ (except for values of k that are nonsensically large in proportion to the size of the dataset). We executed ten trials for each such combination and show

$\mathcal{D}$	Metric	k=10		k=20		k=30		k=50		k=100	
N1	ms	200.4	5.8	197.2	4.6	198.4	3.4	203.2	3.2	206.4	6.0
N1	I/Os	384		384		384		384		385	
N2	ms	198.0	2.8	204.4	6.4	206.0	5.1	210.4	4.7	247.6	7.2
N2	I/Os	384		384		384		384		385	
N3	ms	223.6	6.1	240.0	6.0	259.6	4.8	276.4	3.5	384.4	4.4
N3	I/Os	385		385		385		385		385	
N4	ms	203.2	5.6	210.0	13.6	211.6	4.0	227.2	5.3	280.0	5.7
N4	I/Os	385		384		384		385		385	
N5	ms	198.4	5.4	197.6	7.4	206.8	3.3	219.6	4.8	296.4	4.8
N5	I/Os	385		384		384		384		385	
M	ms	40.4	4.8	41.2	5.0	39.6	4.8	42.0	4.7	N/A	
M	I/Os	44		44		44		44		N/A	
A	ms	15.2	4.1	20.4	3.0	30.0	4.3	N/A		N/A	
A	I/Os	13		13		13		N/A		N/A	
W	ms	14.4	2.8	16.0	4.6	16.0	1.9	N/A		N/A	
W	I/Os	16		16		16		N/A		N/A	

Table 3.4: *Experiment results: construction cost.* The cost, given both in milliseconds and I/Os, to build $\mathcal{C}_k$. For each cell, the value given on the left is the mean of ten trials and the value on the right is the standard deviation (if applicable). "N/A" indicates values of k not run.

$\mathcal{D}$	Alg	k=10	k=20	k=30	k=50	k=100
N1	W	13	13	13	201	203
N1	V	13	13	13	201	203
N1	C	13	13	13	13	13
N2	W	13	15	20	206	325
N2	V	13	201	203	206	325
N2	C	13	13	13	13	13
N3	W	13	25	215	425	871
N3	V	202	212	216	426	1,055
N3	C	13	13	13	13	14
N4	W	174	310	690	1,042	3,456
N4	V	360	867	1,432	2,419	4,647
N4	C	13	13	13	13	14
N5	W	578	1,153	1,788	2,539	4,538
N5	V	1,312	2,593	3,327	5,336	9,721
N5	C	13	13	13	13	14
M	W	267	382	479	716	N/A
M	V	270	398	496	733	N/A
M	C	11	11	11	11	N/A
A	W	42	53	59	N/A	N/A
A	V	60	60	60	N/A	N/A
A	C	10	10	10	N/A	N/A
W	W	10	10	10	N/A	N/A
W	V	10	10	10	N/A	N/A
W	C	10	10	10	N/A	N/A

Table 3.5: *Experiment results: I/O query cost.* The cost, measured in I/Os, to execute the query batch. I/O is consistent across trials, so only one value is reported per dataset-algorithm combination. "N/A" indicates values of k not run.

$\mathcal{D}$	Alg	k=10	k=20	k=30	k=50	k=100
N1	W	$428,384$		constant		
N1	V	$8,572$		constant		
N1	C	$1,224$	$1,252$	$1,308$	$1,868$	$2,244$
N2	W	$428,384$		constant		
N2	V	$7,812$		constant		
N2	C	$1,708$	$2,260$	$3,084$	$3,420$	$5,084$
N3	W	$428,384$		constant		
N3	V	$174,820$		constant		
N3	C	$2,220$	$3,116$	$3,972$	$4,780$	$6,596$
N4	W	$428,384$		constant		
N4	V	$96,100$		constant		
N4	C	$1,764$	$2,172$	$2,868$	$3,716$	$5,428$
N5	W	$428,384$		constant		
N5	V	$185,692$		constant		
N5	C	$1,748$	$1,980$	$2,444$	$3,676$	$6,100$
M	W	$100,724$		constant		N/A
M	V	$107,260$		constant		N/A
M	C	$1,316$	$1,684$	$1,996$	$2,204$	N/A
A	W	$8,724$	constant		N/A	N/A
A	V	$7,832$	constant		N/A	N/A
A	C	$1,772$	$1,804$	$1,900$	N/A	N/A
W	W	$7,984$	constant		N/A	N/A
W	V	$6,872$	constant		N/A	N/A
W	C	$1,844$	$2,260$	$2,540$	N/A	N/A

Table 3.6: *Experiment results: memory usage.* The main memory footprint, given in Bytes, to execute one query. Memory usage is consistent across trials, so only one value is reported per dataset-algorithm combination. "N/A" indicates values of k not run.

$\mathcal{D}$	Desc.	k=10	k=20	k=30	k=50	k=100
N1	$\mathcal{C}_k$	8	8	12	45	70
N1	$\mathcal{H}$	5	6	5	9	6
N1	Disk	152	156	216	760	1,148
N2	$\mathcal{C}_k$	37	69	118	141	244
N2	$\mathcal{H}$	5	10	15	11	13
N2	Disk	616	1,148	1,952	2,302	3,960
N3	$\mathcal{C}_k$	64	118	172	220	337
N3	$\mathcal{H}$	15	19	18	23	16
N3	Disk	1,088	1,968	2,282	3,616	5,460
N4	$\mathcal{C}_k$	37	63	105	160	261
N4	$\mathcal{H}$	12	11	14	10	22
N4	Disk	644	1,056	1,740	2,604	4,268
N5	$\mathcal{C}_k$	38	51	79	154	303
N5	$\mathcal{H}$	8	11	13	17	22
N5	Disk	644	864	1,320	2,536	4,940
M	$\mathcal{C}_k$	13	35	53	68	N/A
M	$\mathcal{H}$	4	6	9	5	N/A
M	Disk	228	588	888	1,112	N/A
A	$\mathcal{C}_k$	40	43	50	N/A	N/A
A	$\mathcal{H}$	7	5	3	N/A	N/A
A	Disk	672	712	816	N/A	N/A
W	$\mathcal{C}_k$	42	69	85	N/A	N/A
W	$\mathcal{H}$	12	10	13	N/A	N/A
W	Disk	724	1,148	1,416	N/A	N/A

Table 3.7: *Experiment results: data structure size.* The size of our data structure, measured in terms of the number of vertices ($\mathcal{C}_k$), number of vertices only on the convex hull ($\mathcal{H}$), and disk space in Bytes. "N/A" indicates values of k not run.

the mean and standard deviation over each set of trials. The Chester et al. algorithm (C) performs constantly with respect to both k and the size of the dataset (call it n), and fastest on all datasets. The Wang et al. algorithm (W) performs linearly with respect to k and without clear trend with respect to n. It outperforms the Vlachou et al. algorithm (V) on all but datasets N1, M, and W. The Vlachou et al. algorithm (V) appears to perform linearly with respect to both k and n.

Table 3.4 discloses the cost of constructing the contour data structure using the preprocessing algorithm of Chester et al. We show the cost both in terms of real time (ms) and I/Os. On all datasets, the wall time grows slowly with k and linearly with n. The I/O cost stays roughly constant.

Table 3.5 gives the number of reads issued while processing the bath of queries for each combination of dataset, algorithm, and k. In all cases, Chester et al. (C) has the fewest I/Os and Vlachou et al. (V) has the most. No pattern emerges with respect to n for any algorithm. With respect to k, Chester et al. (C) is near-constant; both Wang et al. (W) and Vlachou et al. (V) increase I/Os monotonically with k, but according to no clear relationship.

Table 3.6 reports memory usage for each combination of dataset, algorithm, and k. The implementation of Wang et al. (W) has a memory footprint that is constant with respect to k and a direct linear function of n. That of Vlachou et al. (V) has a memory footprint that is also constant with respect to k but does not exhibit a clear relationship with respect to n. For Chester et al. (C), there is linear growth with respect to k but apparent independence of n.

Finally, Table 3.7 reports the size of the contour produced for each dataset at each value of k. Note that these values are independent of any algorithm; they are strictly related to the number of tuples in the dataset at rank k for some tuple. For each trial, we measure the number of vertices in the contour and on the convex hull of the contour and also the disk space consumed by the data structure once written. The size of the convex hull is near-constant, between 3 and 23 vertices. The size of the entire contour, and, correspondingly, its requisite disk space, grows at a slow, linear relationship with k. No pattern emerges with respect to n.

3.4.3 Discussion

Figs. 3.1-3.3 summarize some key points from the results in Sect. 3.4.2. The leftmost charts illustrate trends with respect to k, using dataset N3 (the worst case) as the example. The

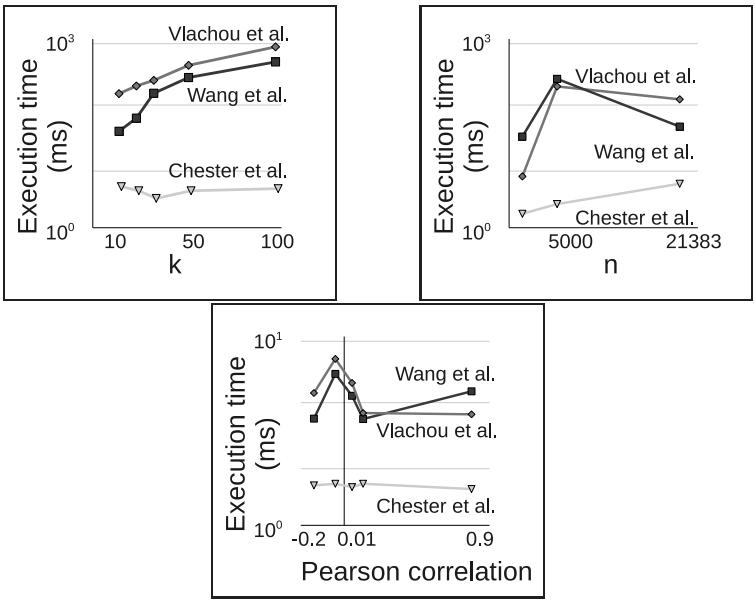


Figure 3.1: *Query execution time for the three algorithms.* On the left, time is a function of k and the dataset is N3, the worst-case dataset used in the experiments. In the middle, time is a function of n, the datasets are W, M, and N3, and $k = 10$, the most typical value used in practice. On the right, time is a function of Pearson correlation and the five database datasets are used. Again, $k = 10$. On all charts, the y-axis is logarithmic. Chester et al. is typically an order of magnitude faster than the other two algorithms.

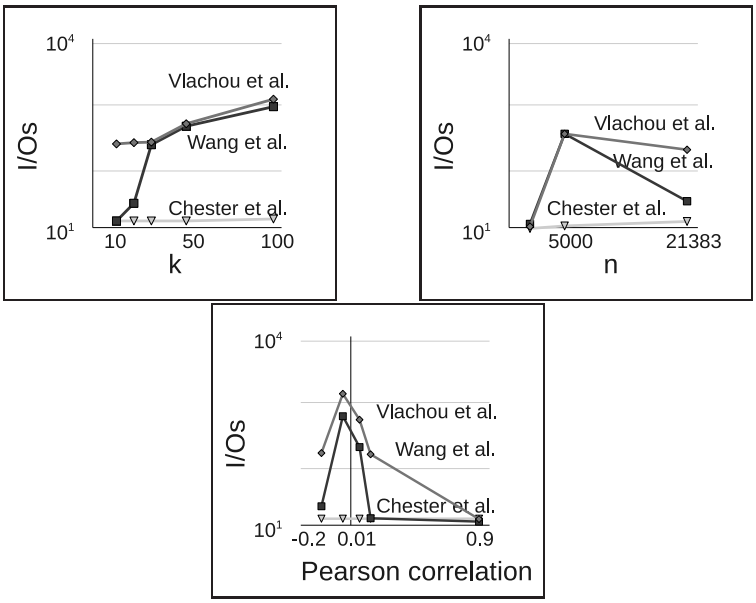


Figure 3.2: *I/O cost for the three algorithms*. On the left, I/Os are a function of k and the dataset is N3. In the middle, I/Os are a function of n, the datasets are W, M, and N3, and $k = 20$, to better illustrate the trends. On the right, I/Os are a function of Pearson correlation and the five database datasets are used. Again, $k = 20$. On all charts, the y-axis is logarithmic. These charts largely explain those in Fig. 3.1.

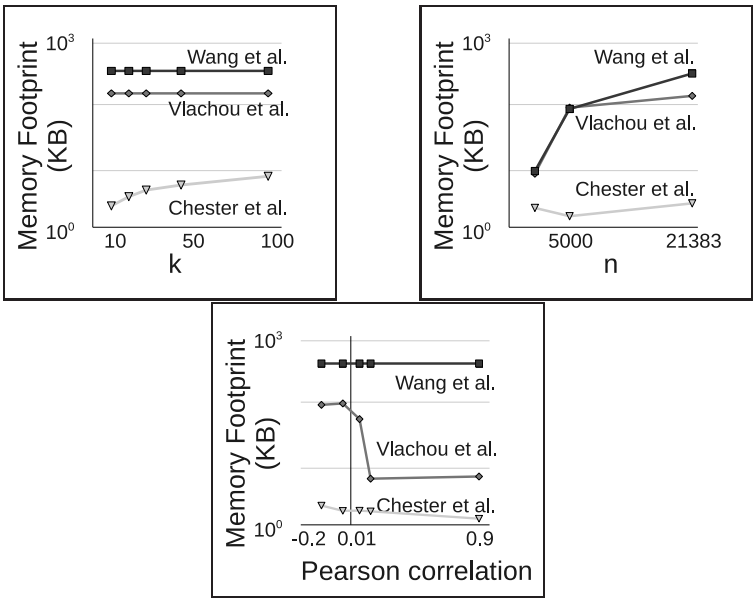


Figure 3.3: *Memory footprint for the three algorithms*. On the left, memory use is a function of k and the dataset is N3. In the middle, memory use is a function of n, the datasets are W, M, and N3, and $k = 10$. On the right, memory use is a function of Pearson correlation and the five database datasets are used. Again, $k = 10$. On all charts, the y-axis is logarithmic. Vlachou et al. outperforms Wang et al. in terms of space.

middle charts illustrate trends with respect to n, fixing $k = 10$ (the most common value in practice), or $k = 20$ if the trends aren't clear at $k = 10$, and using datasets W, M, and N3 (again, the worst case of the five basketball datasets). Finally, the rightmost charts illustrate trends with respect to correlation, fixing $k = 10$ or $k = 20$ and $n = 21,383$. From these figures and results, Chester et al. (C) clearly outperforms Wang et al. (W) and Vlachou et al. (V). Furthermore, from Table 3.4, it takes less than half a second to construct the index to enable the technique. Meanwhile, V and W tend to be a trade-off between space and time.

W always requires more memory than V (Table 3.6 and Fig. 3.3). This occurs because, upon initialization, we allocate a large array with one counter for every possible one of the $n + 1$ segments that are created. Hence the constant memory footprint. It is possible that, at the cost of efficiency, this could be improved with more dynamic memory allocation. Or, a potential research avenue is to investigate compression techniques for the array, since all the counter values are within the range $[1, k]$.

On the other hand, V always requires at least as many I/Os as W (Table 3.5 and Fig. 3.2)--–and is therefore typically slower (Table 3.3 and Fig. 3.1). This occurs because V has a weaker *early-termination* condition than W. To terminate early with a null result, V requires that at least k tuples must pareto-dominate q. W, on the other hand, can detect combinations of tuples that together dominate q, even when neither themselves do.

The three cases in which V performs faster than W correspond exactly to the three datasets with the highest Pearson correlation coefficient on the top 1% of the dataset (Table 4.3). Thus we infer that V is very strong on correlated data, likely because a) the data exhibits a stronger ordering with respect to pareto-dominance, and b) the plane sweep needs process fewer rank inversions. We also conclude that the correlation in the top 1% of a dataset is a much better predictor of performance than the correlation across the entire dataset. This is intuitive because these are the tuples most likely to affect the solution; many of the bottom 99% of tuples are ranked below k for all queries and can be pruned. Table 3.7 and Fig. 3.4 indicate that the contour, at least on the larger datasets, consists of fewer than 1% of the tuples, even for high k and high anticorrelation.

All three algorithms scale more elegantly with respect to n–typically constant or linear-–than with respect to k, which introduces big jumps in I/O or wall-time cost. It is important to consider the context for the queries. In many cases, k will not grow very high. For example, consider dataset A, where $k = 10i$ represents a result being pushed to the i'th page for the user. Conversely, consider dataset M, wherein being ranked among the top hundred threads is noteworthy.

Turning to our algorithm, specifically, we note that our index required 10-20μs to an-

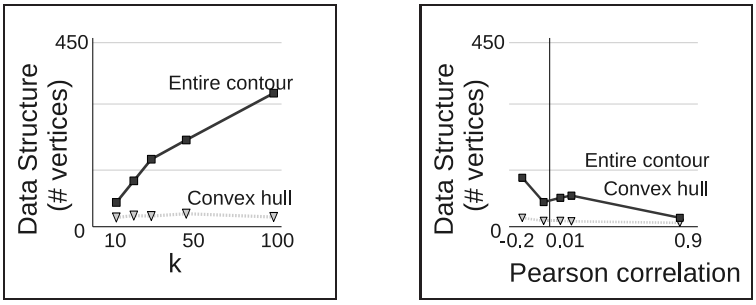


Figure 3.4: *Data structure size*. On the left, the number of data structure vertices is a function of k and the dataset is N3. On the right, size is a function of Pearson correlation and the five database datasets are used. For clarity, $k = 20$. On both charts, the y-axis is *linear*. The size of the convex hull stays roughly constant in contrast to the linear growth of the entire contour with respect to k and anticorrelation .

swer a query on the larger datasets and only the 13 I/Os necessary to load the ≤ 5kB data structure to memory and read the query. The index constructs in well under half a second, often faster than the batched query time for the other two algorithms. The disk space for the contour is surprisingly low. At the largest, it is roughly 5kB, so creating *thousands* of contours on disk for various values of k and combinations of attributes would require mere megabytes.

So, why is our technique so effective? All three algorithms must read the data (although the other two have early-termination conditions). But our preprocessing algorithm only does it once, preferably offline, and certainly in advance of the query. Then, at query time, the index must make a comparison to ≤ 337 points rather than all of $\mathcal{D}$. Naturally, the three techniques were all designed under different assumptions. But if indexing is possible, unmistakably, our technique uses less memory, issues fewer I/Os, and is complete sooner.

3.5 Generalizations

The two dimensional monochromatic reverse top-k query is itself a very interesting and practical problem. Nonetheless, we mention a few generalizations of our work here.

3.5.1 Queries as Existing Points

Throughout this paper, we have assumed that $q \notin \mathcal{D}$. Indeed, however, our indexing structure can account for both the cases, whether $q \in \mathcal{D}$ or not. For our construction algorithm (Algorithm 4), q would be treated just like any other point. The query algorithm (Algorithm 5) would only be affected if q lies exactly on $\mathcal{P}_k$, in which case the return statement on line 4 of the algorithm should return the entire line segment of intersection, rather than just a point.

3.5.2 Bichromatic Reverse Top-k Queries

Our index also straight-forwardly applies to solving *bichromatic* reverse $\text{top-}k$ queries, also introduced by Vlachou et al. [46]. In the bichromatic version, not infinitely all vectors of $\mathbb{R}^2$ are of interest, only those included in an input relation W.

To answer a bichromatic query, one could use our index to find the monochromatic intervals (at most two, by Proposition 3.2.8), and then, for each $w \in W$, determine whether w is in either interval. This requires time $\mathcal{O}(lg\ n + |W|)$, since whether a weight vector lies

in an interval can be established in constant time, and there are most two such intervals to check for each vector.

3.5.3 Higher Dimensions

Many of the concepts introduced in this paper, such as the k-polygons, generalize quite readily to higher dimensions (i.e., to k-polytopes). The analogous challenge to determining the intersection of a query's nullspace with a k-polygon is to intersect a halfplane with a k-polytope. Consequently, the challenge is twofold in higher dimensions: first, to calculate the k-polytope, and, second, to efficiently intersect the translated query nullspace with that k-polytope. The *doubly-connected edge list* is an effective means of representing arrangements and also polytopes in higher dimension. It permits determining the intersection of a halfplane with the interior of a polytope in time linear in the size of the polytope.

An efficient means of creating the polytope would be to simultaneously compute the first k polytopes using an incremental algorithm that updates each polytope for each line. This is still quite efficient, because k is a small constant. At the conclusion of the construction algorithm, the first $k-1$ polytopes could be discarded.

3.6 Bibliographic Notes

Reverse top-k queries, introduced by Vlachou et al. [46,47], are an example of the growing field of Reverse Data Management [35] and have attracted some attention in the past couple years [47, 50, 55]. The work in this chapter is the first query-agnostic work a condensed version of which originally appeared at DASFAA 2013 [12] and an expanded version of which is under revision for an international journal [13]. *Monochromatic* reverse top-k queries, the subject of study here, find application in a number of related problems related to decision support [11, 48], and we detailed their existing algorithms thoroughly in the previous section.

Our approach is among the *layer-based techniques*, which are quite popular among the traditional top-k literature [9, 15, 26, 57] and are based either on the Skyline Operator [6] (as in [9,57]) or a duality transform such as that used in [15] or, in our case, that introduced in [10]. The discovery of a k-contour arises from the application of this duality transform and layer-based approach. Our approach is based on arrangements, a central concept in computational geometry, surveyed nicely by Sharir [41]. The *de facto* standard for representing arrangements is the *doubly connected edge list*, which is detailed quite well in the

introductory text of de Berg et al. [5].

Our analysis of the arrangement is centred around data depth and depth contours. Within Statistics, data depth is a well studied approach to generalising concepts like *mean* to higher dimensions and a number of different depth measures were recently evaluated against each other by Hugg et al. [21]. Top-k rank depth is similar to arrangement depth, which is investigated by Rousseeuw and Hubert [39], particularly with regard to bounding and algorithmically computing the maximum depth of a point within an arrangement. Throughout the paper, we assume only the positive quadrant is of interest in order to simplify discussion and make more direct comparisons to related work. The ideas presented, however, generalize cleanly when the domain of $\mathcal{D}$ is extended to include negative values, an extension shown to be of significant interest by Ranu and Singh [38]. It is important to note that in more general domain, we deviate from these other concepts of data depth by setting the face containing the origin, rather than the external face, to have minimal depth and by not ensuring affine equivariance. As a consequence, we cannot make the assertion about connectedness and monotonicity offered by the study of depth contours by Zuo and Serfling [58] without losing generality.

3.7 Final Remarks

In this chapter we studied an index structure to asympotically improve query performance for reverse top-k queries. We approach the problem novelly by representing the dataset as an arrangement of lines and demonstrating that embedded in the arrangement is a critical k-polygon which encodes sufficient knowledge to respond to reverse top-k queries. In particular, we show that by applying the same transformation to the query tuple to produce a query line l_q, we can retrieve the response to the reverse top-k query on q by intersecting l_q with the interior of the k-polygon.

We derive geometric properties of the problem to bound the query cost and size of our data structure as $\mathcal{O}(\log n)$ and $\mathcal{O}(n)$, respectively. We also conduct an experimental analysis to augment our theoretical analysis and demonstrate both that our algorithm significantly outperforms literature and that our data structure requires little disk space.

In the context of our broader objectives in this dissertation, this chapter does quite well, individually targetting both areas of interest. On the one hand, we offer strong results on assessing the relevance of a two-dimensional tuple to preference querying. On the other, we present a new summarization operator for a dataset, the top-k rank contour, which exactly encodes the k-ranked tuple for any possible preferences.

Chapter 4

k-Regret Minimizing Sets

Regret minimizing sets are a recent approach to representing a dataset $\mathcal{D}$ by a small subset R of size r of representative data points. The set R is chosen such that executing any top-1 query on R rather than $\mathcal{D}$ is minimally perceptible to any user. However, such a minimally perceptible subset R may not exist, even for modest sizes, r. In this chapter, we introduce the relaxation to k-regret minimizing sets, whereby a top-1 query on R returns a result imperceptibly close to the top-k on $\mathcal{D}$.

We show that, in general, with or without the relaxation, this problem is NP-hard. For the specific case of two dimensions, we give an efficient dynamic programming, plane sweep algorithm based on the ideas from the previous chapter in order to find an optimal solution. For arbitrary dimension, we give an empirically effective, greedy, randomized algorithm based on linear programming. With these algorithms, we can find subsets R of much smaller size that better summarize $\mathcal{D}$, using small values of k larger than 1.

Summary of Chapter 4 Contributions

After introducing *k-regret minimizing sets*, we focus on two broad questions in this chapter: (how) can one compute a k-regret minimizing set? We use a diverse toolkit to resolve these questions, including geometric duality, plane sweep, dynamic programming, linear programming, randomization, and reduction. In particular, we:

- generalize *regret ratio*, a top-1 concept, to *k-regret ratio*, a top-k concept, for quantifying how well a subset of a dataset appeals to users (Section 4.1);
- resolve a conjecture by Nanongkai et al. [37], that finding 1-regret minimizing sets is an NP-hard problem and extend the result for k-regret minimizing sets (Section 4.2);

Symbol	Definition
$\mathcal{D}$	The input, a set of points/tuples/vectors
n	$\lvert\mathcal{D}\rvert$, the number of points in $\mathcal{D}$
d	The dimension of the problem
k	A threshold for the rank of an interesting point of $\mathcal{D}$
p_i	The i'th point in $\mathcal{D}$
p_i^j	The j'th coordinate of p_i
R	A subset of $\mathcal{D}$, meant to be a k-regret minimizing set
r	$\lvert R\rvert$, the number of points in R
$\mathbf{w}$	A vector of attribute weights, presumably supplied by a user
$\mathbf{w}_i$	The i'th coordinate of $\mathbf{w}$
$\mathrm{score}(p_i, \mathbf{w})$	$p_i \cdot \mathbf{w}$
$A^{(k,\mathbf{w})}$	The point $p \in A \subseteq \mathcal{D}$ for which $\mathrm{score}(p, \mathbf{w})$ is k highest
$k\text{-gain}(R, \mathbf{w})$	The best score for any point in R on $\mathbf{w}$
$k\text{-regratio}(R, \mathbf{w})$	The ratio of the best score in R on $\mathbf{w}$ vs. the best score in $\mathcal{D}$
$k\text{-regratio}(R)$	The supremum of k-regret ratios on R for all $\mathbf{w}$
$\mathcal{C}_k$	The k-contour, as defined in Chapter 3

Table 4.1: Table of repeatedly used notation for Chapter 4

- introduce an $\mathcal{O}(n^2 r)$ plane sweep, dynamic programming algorithm to compute the size-r k-regret minimizing subset S of a two-dimensional dataset $\mathcal{D}$ for any k (Section 4.3); and

- introduce a randomized greedy linear programming algorithm for the NP-hard case of arbitrary dimension (Section 4.4) that we show performs very well on experiments (Section 4.5).

4.1 Preliminaries

Dataset summarization is an important research challenge, and, as argued in Section 1.3, although the task has been around at least as long as the skyline operator [6], it is still of active research. Here, we introduce *k-regret*, a generalisation of *regret* [37], capturing how far from a k'th "best" tuple is the "best" tuple in a subset.

Throughout this chapter, we assume a dataset $\mathcal{D}$ of n d-dimensional points scaled in each dimension to the range $[0, 1]$. We denote the i'th point by p_i and the j'th coordinate of p^i by p_i^j. We are particularly interested in evaluating a subset $R \subset \mathcal{D}$ as an approximation to $\mathcal{D}$. We consider "linear scoring semantics" as in the previous chapters, wherein a user specifies a vector of real-valued attribute weights, $\mathbf{w} = \langle \mathrm{w}_0, \ldots, \mathrm{w}_{d-1} \rangle$ and a point $p \in \mathcal{D}$

has a score on **w**: $\text{score}(p_i, \mathbf{w}) = \sum_{j=0}^{d-1} p_i^j \mathrm{w}_j$. If one breaks ties arbitrarily, $\mathcal{D}$ can be sorted with respect to **w** in descending order of score, producing a list $(\mathcal{D}^{(1,\mathbf{w})}, \ldots, \mathcal{D}^{(n,\mathbf{w})})$. That is to say, we denote by $\mathcal{D}^{(k,\mathbf{w})}$ the point $p \in \mathcal{D}$ with the k'th highest score, and refer to it as the k-ranked point on **w**.

Then, for each vector of user weights, **w**, we define the k-gain of a subset $R \subseteq \mathcal{D}$, denoted $k\text{-gain}(R, \mathbf{w})$ as the score of the k-ranked point:

Definition 4.1.1 (k-gain)**.**

$$k\text{-gain}(R, \boldsymbol{w}) = \text{score}(R^{(k,\boldsymbol{w})}, \boldsymbol{w}).$$

Recalling the example of Table 1.2, consider $\mathcal{D}_{\text{nba}}$, and the subset of $\mathcal{D}$ given by $R = \{\text{James}, \text{Wade}, \text{Stoudemire}\}$ and $\mathbf{w} = \langle .5, .5, 0, 0\rangle$. Then $R^{(2,\mathbf{w})} = \text{Stoudemire}$ and the 2-gain is 0.447. Also, $1\text{-gain}(R, \mathbf{w}) = 0.522$ and $3\text{-gain}(R, \mathbf{w}) = 0.230$. One can verify this by calculating that on **w** the score for James is 0.522, for Wade is 0.230, and for Stoudemire is 0.447.

With respect to a given vector of user weights, **w**, given a subset $R \subseteq \mathcal{D}$, we define the distance of R from the k'th best choice in $\mathcal{D}$ as the percentage of the k-gain on $\mathcal{D}$:

Definition 4.1.2 (k-regret ratio)**.** *Given a subset $R \subseteq \mathcal{D}$ and a vector of weights, **w**, the k-regret ratio is:*

$$k\text{-regratio}(R, \boldsymbol{w}) = \frac{\max(0, k\text{-gain}(\mathcal{D}, \boldsymbol{w}) - 1\text{-gain}(R, \boldsymbol{w}))}{k\text{-gain}(\mathcal{D}, \boldsymbol{w})}.$$

Note that the ratio must fall in the range $[0, 1]$. Take this time the example of $R = \{\text{James}, \text{Wade}, \text{Bryant}\}$ and $\mathbf{w} = \langle .5, .5, 0, 0\rangle$. Here, $2\text{-regratio}(R, \mathbf{w}) = \frac{0.522 - 0.522}{0.522} = 0$, indicating that a user specifying **w** will find in R his 2-ranked point of $\mathcal{D}_{\text{nba}}$; however, $1\text{-regratio}(R, \mathbf{w}) = \frac{0.717 - 0.522}{0.717} = 0.272$, indicating that a user specifying weights **w** will be 27% 'disappointed' if instead expecting his/her top choice (namely, Durant).

Extending this definition beyond one user weight vector, the *maximum* k-regret ratio for a subset $R \subseteq \mathcal{D}$ is the largest value of k-regratio across all weight vectors $\mathbf{w} \in [0, 1]^d$.

Definition 4.1.3 (maximum k-regret ratio)**.** *Let $\mathcal{L}$ denote all vectors in $[0, 1]^d$. Then the* maximum k-regret ratio *for a subset $R \subseteq \mathcal{D}$ is:*

$$k\text{-regratio}(R) = \sup_{\boldsymbol{w} \in \mathcal{L}} k\text{-regratio}(R, \boldsymbol{w}).$$

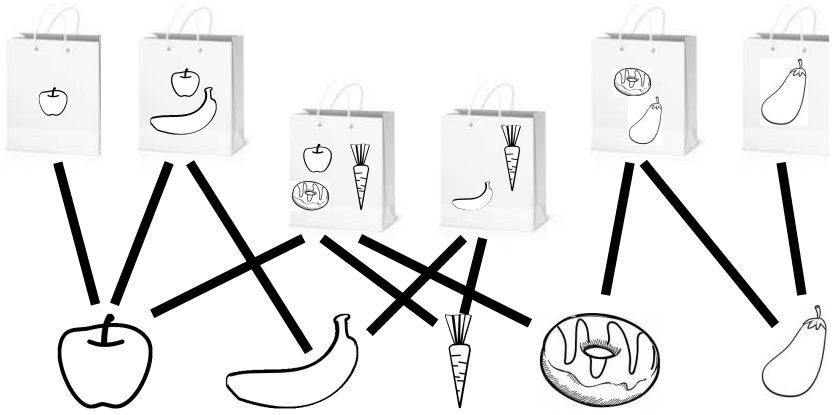


Figure 4.1: *An example of the set cover problem.* Choose at most r of the grocery bags above such that the union of all r grocery bags contains everything in the five-item grocery list below.

The *maximum k-regret ratio* is the largest observable k-regret ratio for any user weight vector. For R as above the 1-regret ratio is maximized for $\langle 0, 1, 0, 0\rangle$, at which the best score obtainable in R is 0.622 and the 1-regret ratio is $(1.000 - 0.622)/1.000 = 0.378$.

This brings us to the primary objective in this paper, to produce a fixed-cardinality k-*regret minimizing set*. That is, given an integer r and a dataset $\mathcal{D}$, the goal is to discover a subset $R \subseteq \mathcal{D}$ of size r that achieves the minimum possible maximum k-regret ratio.

Definition 4.1.4 (k-regret minimizing set)**.** *A k-regret minimizing set of order r on a dataset $\mathcal{D}$ is:*

$$R_r(\mathcal{D}) = \text{argmin}_{R \subseteq \mathcal{D}, |R|=r} k\text{-regratio}(R).$$

Definition 4.1.5 (kRMS)**.** *Given a set $\mathcal{D}$ of n points in d dimensions and an integer r, return a k-regret minimizing set of size r on $\mathcal{D}$.*

Note that if $k = 1$ these definitions reduce to analagous ones introduced by Nanongkai et al. [37]. We refer to that problem as 1RMS.

4.2 Regret Minimization is Hard

We begin by resolving a conjecture by Nanongkai et al. [37] that 1RMS is hard.

id	a	b	c	d	e
d_0	1	0	0	0	0
d_1	0	1	0	0	0
d_2	0	0	1	0	0
d_3	0	0	0	1	0
d_4	0	0	0	0	1
p_0	.2	0	0	0	0
p_1	.2	.2	0	0	0
p_2	.2	0	.2	.2	0
p_3	0	.2	.2	0	0
p_4	0	0	0	.2	.2
p_5	0	0	0	0	.2

Table 4.2: *An instance of 1RMS corresponding to the set cover problem in Fig. 4.1.* Each axis point d_0 to d_4 corresponds to an element of $\mathcal{U}$ and each mapped point p_0 to p_5 corresponds to an element of T. Any subset of r points with a maximum regret of $1-1/|\mathcal{U}| = 0.8$ is a solution to the original set cover problem. If no such subset exists, no solution exists in Fig. 4.1.

Theorem 7 (Hardness of 1RMS)**.** $1\mathrm{RMS}$ *is NP-Hard.*

We prove Theorem 7 by means of a reduction from the SET-COVER decision problem. The SET-COVER decision problem is, from a finite collection of subsets T of a finite universe $\mathcal{U}$, to determine if one can select r subsets from T so that every element of $\mathcal{U}$ is included. SET-COVER was shown to be NP-Complete by Karp [25].

Problem Definition 2. [SET-COVER] Given a finite universe $\mathcal{U}$, a set T of subsets of $\mathcal{U}$, and an integer r, is there a size r subset $S = \{s_0, \ldots, s_{r-1}\}$ of T such that $\bigcup_{s_i \in S} s_i = \mathcal{U}$?

At a high level, we reduce an instance of SET-COVER=$(\mathcal{U}, T, r)$ to an instance of 1RMS=$(\mathcal{D}, r)$ as follows. We construct $\mathcal{D}$ with $n = |\mathcal{U}|+|T|$ points in $d = |\mathcal{U}|$ dimensions. The first $|\mathcal{U}|$ "axis" points correspond to elements of $\mathcal{U}$ and the last $|T|$ "mapped" points of $\mathcal{D}$ correspond to elements of T. The coordinate values are chosen appropriately, higher for axis points than for mapped points, so that a 1RMS solution on $\mathcal{D}$ then maps back to a solution of the SET-COVER problem.

The formal details follow shortly, but first we illustrate the reduction with an example of SET-COVER in Fig. 4.1 and a corresponding $|\mathcal{U}|$-dimensional 1RMS instance in Table 4.2. For each element of $\mathcal{U}$, one creates a unique axis point (d_0 to d_4 in Table 4.2). For example, "banana" becomes $d_1 = (0, 1, 0, 0, 0)$ and "eggplant" becomes $d_4 = (0, 0, 0, 0, 1)$. For each subset s_i in T, one creates an additional point (p_0 to p_5 in Table 4.2). Each coordinate p_i^j

is 0 if the j'th element is not in s_i and is $|\mathcal{U}|^{-1}$ if it is. For example, $\{\text{banana}, \text{carrot}\}$ becomes $p_3 = (0, .2, .2, 0, 0)$. This forms the dataset for 1RMS.

To be more precise, we build a reduction from the decision SET-COVER problem defined in Problem Definition 2. Given an instance $\mathcal{I}_{\text{SC}} = (\mathcal{U}, T, r)$ of SET-COVER, we produce an instance $\text{I}_{\text{RMS}} = (\mathcal{D}, r)$ of regret minimization as follows. For every element $e_i \in \mathcal{U}$, construct an *axis point* $d_i = (d_i^0, \ldots, d_i^{|\mathcal{U}|-1})$, where $d_i^j = 1$ if $i = j$ and $d_i^j = 0$ otherwise. For every element $s_i \in T$, construct a *mapped point* $p_i = (p_i^0, \ldots, p_i^{|\mathcal{U}|-1})$, where $p_i^j = |\mathcal{U}|^{-1}$ if $e_j \in s_i$ and $p_i^j = 0$ otherwise. Let $\mathcal{D}$ be the set of all axis and mapped points so constructed. Let I_{RMS} be the instance of regret minimization produced with $\mathcal{D}$ and the same value of r as in $\mathcal{I}_{\text{SC}}$. This completes the reduction, which runs in polynomial time (Proposition 4.2.1).

Proposition 4.2.1. *The reduction from SET-COVER to 1RMS takes $\mathcal{O}(|\mathcal{U}|(|\mathcal{U}| + |T|))$ time for the construction of I_{RMS}.*

Proof [of Theorem 7] We now use the reduction described above to prove Theorem 7. We show that there are only three possible maximum regret ratios that can be produced by an instance or 1RMS constructed with this reduction (Lemma 4.2.2), and a yes or no decision for $\mathcal{I}_{\text{SC}}$ each corresponds exactly to those cases (Lemma 4.2.3).

Lemma 4.2.2. *Given $\text{I}_{\text{RMS}} = (\mathcal{D}, r)$, any subset $R \subseteq \mathcal{D}$ has $k\text{-regratio}(R) \in \{0, 1, 1 - d^{-1}\}$.*

Proof First, note that $\forall \mathbf{w}$, the top-ranked point is an axis point. Precisely, if j is the largest coordinate of $\mathbf{w}$, then $\mathcal{D}^{(1,\mathbf{w})} = d_j$, because $\forall p_i \in \mathcal{D}, \sum_{h=0}^{d-1} w_h p_i^h \leq w_j$. (The coordinate values of each p_i were chosen specifically to guarantee this condition.)

Second, note that for any subset $R \subseteq \mathcal{D}$, either 1) all axis points are in R; or 2) $\exists d_j \notin R$, where d_j is some j'th axis point. This second case refines further: 2a) $\exists d_j \notin R, \forall p_i \in R, p_i^j = 0$; and 2b) $\forall d_j \notin R, \exists p_i \in R$ with $p_i^j = d^{-1}$. We analyse each of the three cases in order.

In case 1), $k\text{-regratio}(R) = 0$, because every top-ranked point is an axis point, and every axis point is in R. In case 2a), let d_j be an axis point fulfilling the case condition. The weight vector $\mathbf{w}$ formed by setting the j'th coordinate to 1 and every other coordinate to 0, establishes $k\text{-regratio}(R, \mathbf{w}) = 1$, the maximum possible value for a k-regret ratio, so $k\text{-regratio}(R) = 1$. In the final case, 2b), consider a weight vector $\mathbf{w}$ that maximizes the expression $1 - \frac{\mathbf{w} \cdot p_i}{\mathbf{w} \cdot d_j}$, for some $d_j \notin R$. Clearly, the j'th coordinate of $\mathbf{w}$ must have some non-zero value, c, or else $\mathbf{w} \cdot d_j = 0$. But any non-zero value on any other coordinate of $\mathbf{w}$ can

only increase the numerator without increasing the denominator, since the denominator, $\mathbf{w} \cdot d_j$, has only the one non-zero coordinate. So, $\mathbf{w}$ must be constructed as in case 2a). Then, $k\text{-regratio}(R, \mathbf{w}) = 1 - \frac{cp_i^j}{c} = 1 - d^{-1}$. So, in conclusion, the maximum regret ratio must take on one of the three values: $\{0, 1, 1 - d^{-1}\}$. □

Lemma 4.2.3. *An instance $\mathcal{I}_{\text{SC}}$ has a set cover of size r if and only if the corresponding instance I_{RMS} has a 1-regret minimizing set of size r with 1-regret ratio < 1.*

Proof Without loss of generality, assume that we do not have a trivial case of set cover where either $r \geq |\mathcal{U}|$ or $\exists u \in \mathcal{U} : \forall s_i \in S, u \notin s_i$, since both cases are easily resolved in polynomial time. Then:
If: Let R be a solution to I_{RMS} with a 1-regret ratio of $1 - d^{-1}$. Then $\mathcal{I}_{\text{SC}}$ has a set cover, namely the one containing the set s_i for every mapped point $p_i \in R$ and a set $s \in S$ containing j for each axis point $d_j \in R$.
Only if: Assume that there is a set cover S of size r on $\mathcal{I}_{\text{SC}}$. Every subset $s \in S$ has a corresponding mapped point $p_i \in \mathcal{D}$ with $p_i^j = d^{-1}$ for all $j \in s$. So, since S covers every $u \in \mathcal{U}$, then for every dimension j, some p_i in the solution has a non-zero p_i^j. The regret ratio is maximized on the axes; so, the maximum regret ratio is at most $1 - d^{-1}$. But from Lemma 4.2.2, the only other possible value is 0, which corresponds to one of the trivial cases. □

Therefore, we have built a correct polynomial-time reduction from SET-COVER to 1RMS. This completes the proof of Theorem 7. □

To see an example of the case correspondence in Lemma 4.2.3, take any two points p and p' in Table 4.2. There will be some j'th dimension not "covered" by p nor p' and setting $\mathbf{w}$ identical to d_j will produce a regret ratio of 1. For $p = p_2$ and $p' = p_5$, for example, d_1 is one such axis point. Notice, too, that there are no two grocery bags in Fig. 4.1 that cover the grocery list. On the other hand, $R = p_1, p_3, p_4$ has a regret ratio of $1 - d^{-1} = 0.8$ and the set $\{s_1, s_3, s_4\}$ in Fig. 4.1 covers the complete grocery list.

Corollary 4.2.4. *$k\text{RMS}$ is NP-Hard.*

PROOF SKETCH. The reduction proceeds analogously to that of Theorem 7, except that we create k axis points for every element of $\mathcal{U}$ rather than just one. Then, the rank of every mapped point is k, there is no benefit in selecting multiple copies of axis points, and the other details of the proof remain the same.

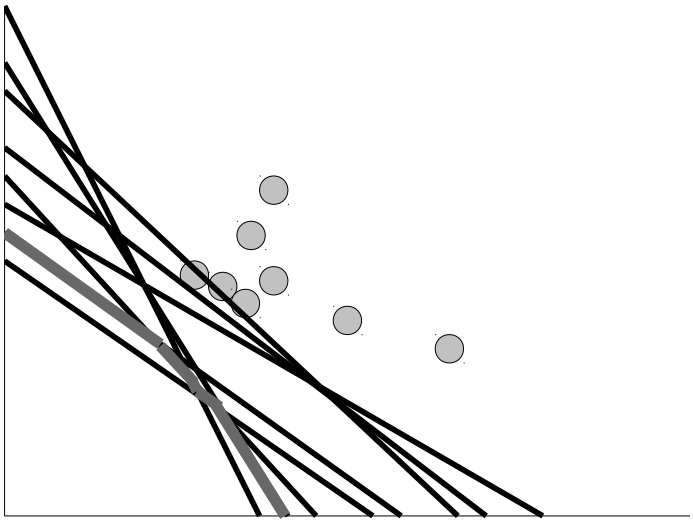


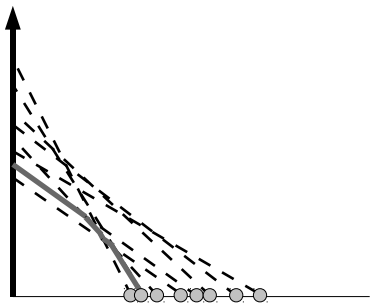
Figure 4.2: *Dual space depiction of Table 1.2.* Shown are the first 2 attributes of the 8 basketball players from Table 1.2, both in primal space (points) and dual space (lines). The thicker, light magenta lines is the top-2 rank contour.

4.3 A Contour View of Regret

The alert reader may have noticed that the hardness result of the previous section holds only for arbitrary dimension. In this section we give the first algorithm to find a k-regret minimizing set in two dimensions, consequently implying that kRMS$\in$P for the case $d = 2$. The main algorithmic insight is showing that kRMS is equivalent in dual space to finding a "convex chain" that is "closest" to a "top-k rank contour." In Section 4.3.1 we formalize this notion; then, in Section 4.3.2, we present the algorithm; and finally, in Section 4.3.3, we prove the algorithm's correctness.

4.3.1 Convex Chains and Contours

Throughout Section 4.3, we will reason in terms of *dual space*. That is to say, all operations on points in $\mathcal{D}$ will instead be performed on lines in a set $\mathcal{L}(\mathcal{D})$, formed as follows. For each point $p_i \in \mathcal{D}$, construct line $l_i \in \mathcal{L}(\mathcal{D})$ as $y = (1 - p_i^0 x)/p_i^1$. For example, $\mathcal{D}_{\text{nba}}$, when projected on just the first two attributes, will be transformed into the dual space lines depicted in Fig. 4.2. An important property of this transform is that two lines l_i and l_j intersect at a point (x, y) exactly when $\text{score}(p_i, \langle x, y\rangle) = \text{score}(p_j, \langle x, y\rangle)$; i.e., p_i and p_j have the same score in primal space on the weight vector $\mathbf{w} = \langle x, y\rangle$. When $\mathcal{D}$ is clear from the context we will use $\mathcal{L}$ to denote $\mathcal{L}(\mathcal{D})$ for notational convenience.



(a) $\mathcal{L}(\mathcal{D}_{\mathrm{nba}})$ after initialisation.

$\mathcal{Q}$

Intersections	*slope of* L
(Stoudemire, Randolph)	2.64
(Bryant, Anthony)	6.07
(Anthony, Stoudemire)	15.39
(Wade, Nowitzki)	35.71

(b) Initial values for $\mathcal{Q}$.

$\mathcal{P}$

End lines	≤ 0	≤ 1
Durant	0.000	0.000
James	0.000	0.000
Wade	0.057	0.057
Nowitzki	0.062	0.062
Bryant	0.080	0.080
Anthony	0.089	0.089
Stoudemire	0.105	0.105
Randolph	0.188	0.188

(c) The initialized $\mathcal{P}$ matrix.

Figure 4.3: *The initialisation of Algorithm 6.* The sweep line L begins at the positive x-axis. The priority queue $\mathcal{Q}$ is populated with intersection points of neighbouring lines that intersect in the positive quadrant, sorted by the order in which L will pass through them. Every entry in the path list $\mathcal{P}$ is originally set to the distance of the corresponding line to the contour along the x-axis.

In the dual space, we are interested in two types of objects and their relationship. The first object we call a *convex chain*. The other, a *top-k rank contour*, we borrow from [12]. Their definitions follow.

Definition 4.3.1 (convex chain)**.** *A* convex chain *of size r is a sequence of line segments, $(s_0, \ldots, s_{r-1})$, entirely in the positive quadrant, where s_0 has an endpoint on the positive x-axis, s_{r-1} has an endpoint on the positive y-axis, the slope of any segment s_i is negative and less than that of s_{i+1}, and any two consecutive line segments have a common endpoint. We call the common endpoints* joints *of the convex chain.*

One can alternatively view a convex chain $(s_0, \ldots, s_{r-1})$ as the lower envelope of the lines $l(s_i)$ obtained by extended each s_i infinitely in both directions. A related concept is that of a *top-k rank contour*.

Definition 4.3.2 (top-k rank contour [12])**.** *Given a set of lines $\mathcal{L}$, the* top-k rank contour*, denoted $\mathcal{C}_k$, is the set of points from each $l_i \in \mathcal{L}$ such that for any point $p \in \mathcal{C}_k$, exactly $k-1$ lines in $\mathcal{L}$ cross the segment $[O,p]$ (ties withstanding).*

As an example, Fig. 4.2 depicts in magenta the top-2 contour, $\mathcal{C}_2$, of $\mathcal{L}(\mathcal{D}_{\text{nba}})$. Whereas a convex chain corresponds to the points on a set of lines closest to the origin, the points on the top-k rank contour are all a 'depth' of k away. Thus, the top-1 rank contour is, itself, a convex chain, but the chains formed at other ranks are not necessarily convex. In general, as shown by Chester et al. [12], the top-k rank contour of n lines can be found in $\mathcal{O}(n \lg n)$ time.

Just before introducing our algorithm in the next subsection, we introduce some short-lived notation to ease the discussion. Consider some fixed vector of user weights, $\mathbf{w} = \langle w_0, w_1 \rangle$ and a line l_i. Let $(a_i w_0, a_i w_1)$ be the point on l_i in the direction of $\mathbf{w}$, and let $\delta_{\mathbf{w}}(O, l_i)$ denote the distance to that point from the origin O . Similarly, let $\delta_{\mathbf{w}}(O, \mathcal{C}_k)$ denote the distance to the point (bw_0, bw_1) on the top-k depth contour in the direction of $\mathbf{w}$ and let $\delta_{\mathbf{w}}(\mathcal{C}_k, l_i) = \max(0, \delta_{\mathbf{w}}(O, l_i) - \delta_{\mathbf{w}}(O, \mathcal{C}_k))$ denote the distance from the contour to the line l_i (or 0 if the distance is negative).

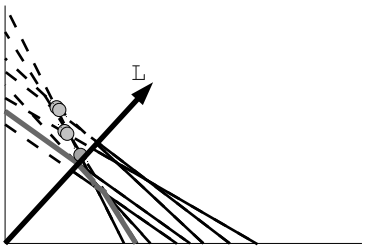
We will say that a line l_i is "closer" to $\mathcal{C}_k$ than another line l_j in the direction $\mathbf{w}$ when the distance ratio $\Delta(l_i) = \delta_{\mathbf{w}}(\mathcal{C}_k, l_i)/\delta_{\mathbf{w}}(O, l_i)$ is less than when computed with l_j: $\Delta)l_j) = \delta_{\mathbf{w}}(\mathcal{C}_k, l_j)/\delta_{\mathbf{w}}(O, l_j)$. We emphasize that the *ratio* is of interest, *not* the difference, because, as we will show in Section 4.3.3, this distance ratio is equivalent to the k-regret ratio on $\mathbf{w}$ in primal space. Finally, the maximum distance ratio from a contour $\mathcal{C}_k$ of an entire convex chain is: $\Delta(\mathcal{C}_k, (s_0, \ldots, s_{r-1})) = \sup_{\mathbf{w}} \min_{s_i} \Delta(l(s_i))$.

4.3.2 An Algorithm for Two Dimensions

Algorithm 6 operates in dual space on the set of lines $\mathcal{L}(\mathcal{D})$. The goal is to find a convex chain of size r within $\mathcal{L}(\mathcal{D})$ whose maximum distance ratio from $\mathcal{C}_k$ is minimized. This gives the k-regret minimizing set $R_r(\mathcal{D})$ by transforming back into points all the lines contributing segments in the convex chain.

Algorithm Description

At a high level, the algorithm is a radial plane sweep *and* dynamic programming algorithm. A sweep line `L` rotates from the positive x-axis to the positive y-axis, tracing out the possible convex chains in $\mathcal{L}$. This is achieved with two data structures: a sorted list of $\mathcal{L}$



(a) $\mathcal{L}(\mathcal{D}_{\text{nba}})$ immediately before processing the darker point, (Stoudemire, Randolph).

	$\mathcal{P}$	
End lines	≤ 0	≤ 1
Durant	0.000	0.000
James	0.00	0.000
Wade	0.057	0.057
Nowitzki	0.062	0.059 (Wade)
Bryant	0.080	0.080
Anthony	0.089	0.084 (Bryant)
Stoudemire	0.105	0.066 (Wade)
Randolph	0.188	**0.083 (Wade)**

(b) Values of $\mathcal{P}$ just after processing (Stoudemire, Randolph). The bold value indicates the unchanged value.

Figure 4.4: *The processing of an event point.* The darker point in (a) is the event point, the intersection of the lines corresponding to Stoudemire and Randolph. The distance along L of $p_{i,j}$ from the contour is 0.034. In this case, the newly discovered length-2 chain, (Stoudemire, Randolph), has cost $\max(0.105, 0.034)$, which does not improve on the value already found for Randolph, 0.083.

and a priority queue $\mathcal{Q}$ containing some of the joints at which lines in $\mathcal{L}$ intersect. Meanwhile, in a matrix $\mathcal{P}$ we keep track of the $n \times r$ best convex chains found so far between the positive x-axis and the sweep line, L. This is the dynamic programming part. The algorithm progresses as L encounters new intersection points throughout its radial sweep; at each, $\mathcal{L}$ and $\mathcal{Q}$ are updated to reflect the new state, and the best convex chains are updated as needed.

Because of its event-driven nature, the crux of the algorithm is in updating the data structures at each intersection (i.e., event) point, which we describe in Section 4.3.2. Algorithm 6 presents the overall framework and Fig. 4.3 illustrates the initialisation of the data structures for the $\mathcal{D}_{\text{nba}}$ example. First, we give a quick description of each data structure.

The set of lines $\mathcal{L}$ is always sorted with respect to the distance from the origin at which they intersect the sweep line L. As L rotates, $\mathcal{L}$ needs to be updated because the sort order may change.

The priority queue $\mathcal{Q}$ contains, sorted by angle, intersection points that have been 'discovered.' An intersection point of lines l_i, l_{i+1} is never discovered until l_i and l_{i+1} are immediate neighbours in $\mathcal{L}$ *and* their intersection point lies between L and the positive y-axis. This discovery process keeps the size of the $\mathcal{Q}$ smaller.

Finally, the array $\mathcal{P}$ contains a cell (i, j) for the best convex chain found up to L that

	$\mathcal{P}$	
End lines	≤ 0	≤ 1
Durant	0.114	0.114
James	0.208	0.208
Wade	0.625	0.625
Nowitzki	0.117	0.059 (Wade)
Bryant	0.566	0.566
Anthony	0.397	0.081 (Wade)
Stoudemire	0.105	**0.000 (Durant)**
Randolph	0.188	**0.000 (James)**

Figure 4.5: *The conclusion of Algorithm 6.* The $\mathcal{P}$ data structure at the conclusion of the plane sweep, when L reaches the y-axis and $\mathcal{Q}$ is empty. Each cell (h, i) of $\mathcal{P}$ contains the optimum solution with i or fewer lines, ending on line l_h. The bold, minimum value in $\mathcal{P}$ gives the two equal solutions for this two-dimensional kRMS problem.

ends in a segment of l_i and contains no more than j joints. Stored in each cell is the convex chain itself *and* the maximum distance ratio of that convex chain from $\mathcal{C}_k$.

The algorithm itself is a traditional plane sweep. Intersection points are popped from $\mathcal{Q}$, the data structures are updated as per Section 4.3.2, and new intersection points are discovered and inserted into $\mathcal{Q}$. For the running basketball example, this is illustrated in Figure 4.4. For intersection points that happen to lay on the contour (since these, too, are intersection points of lines that will eventually be discovered by the plane sweep), we update every cell of $\mathcal{P}$ with new maximum costs. Eventually, $\mathcal{Q}$ is exhausted when L reaches the positive y-axis (see Figure 4.5) and the best convex chain is read from $\mathcal{P}$.

Data structure transitions

We now describe the updates to the three data structures when L is at an arbitrary intersection point of lines l_i and l_j, denoted $p_{i,j}$. For simplicity, assume that only two lines intersect at any given point; it is straight-forward to handle lines not in general position.

$\mathcal{L}$

Because the lines are intersecting, we know they are immediately adjacent in $\mathcal{L}$. We swap l_i and l_j in $\mathcal{L}$ to reflect the fact that immediately after $p_{i,j}$, they will have opposite order as beforehand.

Algorithm 6 Two-dimensional kRMS algorithm

1: **Input**: $\mathcal{D}$; r
2: **Output**: $R \subseteq \mathcal{D}$, with $|R| = r$ and minimum k-regret ratio
3: Compute $\mathcal{C}_k$.
4: Transform set of points $\mathcal{D}$ into set of lines $\mathcal{L}$.
5: Sort $\mathcal{L}$ by ascending x-intercept.
6: Initialize $\mathcal{Q}$ with intersection points of all $l_i, l_{i+1} \in \mathcal{L}$, sorted by angle from positive x-axis.
7: Initialize $\mathcal{P}$ as $n \times r$ matrix, cell (i, j) has path (i) and cost $\delta_{\langle 1,0\rangle}(\mathcal{C}_k, l_i)/\delta_{\langle 1,0\rangle}(O, l_i)$.
8: **while** $\mathcal{Q}$ is not empty **do**
9: Pop top element q off $\mathcal{Q}$.
10: Update data structures $\mathcal{L}, \mathcal{Q}, \mathcal{P}$ as per Section 4.3.2.
11: **end while**
12: Update each cell $(i, r-1)$ of $\mathcal{P}$ with cost at positive y-axis.
13: Find cell $(i, r-1)$ in $\mathcal{P}$ with lowest cost.
14: Init empty set R.
15: **for all** j on convex chain in cell $(i, r-1)$ **do**
16: Add p_j to R
17: **end for**
18: RETURN R

$\mathcal{Q}$

Immediately after $p_{i,j}$, lines l_i and l_j have been swapped in $\mathcal{L}$. So, potentially, two new intersection points are discovered: viz., l_i and his new neighbour (should one exist) and l_j and his new neighbour (again, should one exist). Both these intersection points are added to the appropriate place in $\mathcal{Q}$, provided that they are between L and the positive y-axis. The old point is removed.

$\mathcal{P}$

Of the four paths, $(l_i, l_i), (l_i, l_j), (l_j, l_i), (l_j, l_j)$, through $p_{i,j}$, only three are valid, as illustrated in Figure 4.6. For a line transiting through $p_{i,j}$, such as (l_i, l_i), the convex chain does not change, but the cost is updated to the larger of what the value was before and the distance ratio, $\delta_{\mathsf{L}}(\mathcal{C}_k, l_i)/\delta_{\mathsf{L}}(O, l_i)$. For the convex chain that turns, (l_i, l_j), the cost in cell (j, h) depends on the best route to get to $p_{i,j}$. Specifically, the best convex chain of size h to $p_{i,j}$ is either: 1) the chain incoming on l_j if the cost in cell (j, h) is smaller; or 2) the chain incoming on l_i if the cost in cell $(i, h-1)$ is smaller. Call the smaller cost mc. The cost for cell (j, h) then becomes the larger of mc and the distance ratio, $\delta_{\mathsf{L}}(\mathcal{C}_k, l_i)/\delta_{\mathsf{L}}(O, l_i)$.

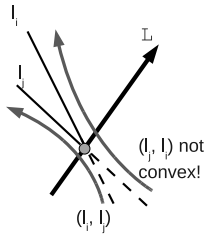


Figure 4.6: *The three legal, convex paths through an intersection point.* Either line l_1 or l_2 could simply pass through. Because an envelope of lines must form a convex chain, on the other hand, only l_2 has the luxury of turning onto l_1. The path (l_2, l_1) requires an "illegal" concave turn.

For rows i and j, each of the r cells is updated in this manner. If $p_{i,j}$ happens to be a vertex of $\mathcal{C}_k$ (i.e., l_i or l_j are sorted k'th in $\mathcal{L}$), then every cell of every row is updated in this manner.

Lemma 4.3.1. *Algorithm 6 finds a k-regret minimizing set of size r for $d = 2$ in $\mathcal{O}(rn^2)$ time with $\mathcal{O}(n^2)$ space.*

PROOF SKETCH. The space comes from storing the dynamic programming matrix, $\mathcal{P}$. The running time is dominated by updating $\mathcal{O}(r)$ cells of $\mathcal{P}$ for each of the $\mathcal{O}(n^2)$ intersection points or by updating all nr cells of $\mathcal{P}$ for the up to n vertices of $\mathcal{C}_k$.

4.3.3 Two-Dimensional kRMS Transformed

Algorithm 6 computes a k-regret minimizing set by discovering a 'best' convex chain, examining the joints of the convex chain and the vertices of $\mathcal{C}_k$. In this section, we prove the sufficiency of that approach, Theorem 8. We first connect convex chains and contours to kRMS in Lemma 4.3.2 below. The lemma equates the distance ratio of a convex chain and contour to the k-regret ratio in primal space. Then, in Lemma 4.3.4, we prove that this ratio must be maximized for any convex chain either at one of its joints or at a vertex of the contour. We make a slight abuse of notation and indicate by a vector of user weights $\mathbf{w} = \langle w_0, w_1 \rangle$ both the vector and the point (w_0, w_1).

Let $V(\mathcal{C}_k)$ denote the vertices of $\mathcal{C}_k$. Also, let $cc = (s_0, \ldots, s_{r-1})$ be a convex chain with joints J that is the lower envelope of a set of lines, $\mathcal{L}' \subseteq \mathcal{L}(\mathcal{D}) = \{l_0, \ldots, l_{r-1}\}$. Say, too, that cc minimizes $\Delta(\mathcal{L}') = \max_{\mathbf{w} \in J \bigcup V(\mathcal{C}_k)} \min_{l_i \in \mathcal{L}'} \delta_{\mathbf{w}}(\mathcal{C}_k, l_i) / \delta_{\mathbf{w}}(O, l_i)$ among all convex chains in $\mathcal{L}$. Algorithm 6 finds such a chain. Then:

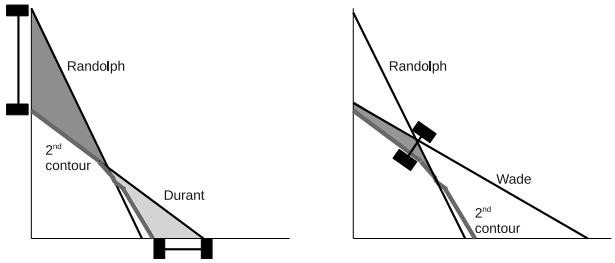


Figure 4.7: An illustration of 2RMS in dual space. Shown left are two different size-1 sets, $\{\mathrm{Durant}\}$ and $\{\mathrm{Randolph}\}$, along with the second contour. The axes are the directions of maximum regret ratio for the respective sets. To the right is shown the (non-optimal) size-2 set $\{\mathrm{Randolph}, \mathrm{Wade}\}$, again with the maximum regret ratio direction indicated.

Theorem 8 (Dual space equivalent of $2d$ kRMS)**.** *For cc as described above, if one transforms each $(y = (1 - p_i^0 x)/p_i^1) \in \mathcal{L}'$ into the point (p_i^0, p_i^1), the set of all such points is the k-regret minimizing set, $R_r(\mathcal{D})$.*

Proof This proof is in three parts. First, in Lemma 4.3.2, we show that k-regret ratio in primal space is equivalent to the distance ratio in dual space, with respect to a given $\mathbf{w}$. Then, in Lemma 4.3.3, we show that for a given set of lines $\mathcal{L}' \subseteq \mathcal{L}$, the distance ratio should always be computed with respect to the lower envelope. Finally, we show in Lemma 4.3.4 that the maximum distance ratio between a contour and a convex chain always occurs at a vertex of one or the other, never in the middle of a line segment.

Lemma 4.3.2. *For any weight vector $\mathbf{w}$ and point $p_i \in \mathcal{D}$ transformed to line $l_i \in \mathcal{L}$:*

$$k\text{-regratio}(\{p_i\}, \mathbf{w}) = \frac{\delta_{\mathbf{w}}(\mathcal{C}_k, l_i)}{\delta_{\mathbf{w}}(O, l_i)}.$$

Proof First note that for any p_i, l_i, and $\mathbf{w}$, $\delta_{\mathbf{w}}(O, l_i) = 1/\mathrm{score}(p_i, \mathbf{w})$. So,

$$\begin{aligned}
\delta_{\mathbf{w}}(\mathcal{C}_k, l_i) &= \delta_{\mathbf{w}}(O, l_i) - \delta_{\mathbf{w}}(O, \mathcal{C}_k) \\
&= \frac{1}{a_i(w_0^2 + w_1^2)} - \frac{1}{b(w_0^2 + w_1^2)} \\
&= \frac{b(w_0^2 + w_1^2) - a_i(w_0^2 + w_1^2)}{a_i b(w_0^2 + w_1^2)^2} \\
&= k\text{-regratio}(\{p_i\})\delta_{\mathbf{w}}(O, l_i)
\end{aligned}$$

□

Lemma 4.3.3. *For a set of lines $\mathcal{L}$ corresponding to a set of points R, given a weight vector* **w***, the distance ratio at the lower envelope of $\mathcal{L}$ gives k-$\mathrm{regratio}(R, \mathbf{w})$.*

Proof Consider a fixed weight vector, **w**. Recall from Definition 4.1.2 that a choice of R minimizes the k-regret ratio with respect to **w** if it maximizes 1-$\mathrm{gain}(R, \mathbf{w})$. Since a line l_i is at a distance of $1/\mathrm{score}(p_i, \mathbf{w})$ from O in the direction **w**, the point p_i with highest score in R will be nearest to O and hence on the lower envelope. □

Lemma 4.3.4. *Consider a contour $\mathcal{C}_k$ and a convex chain of line segments $\mathcal{L}$. The maximum ratio $\Delta(\mathcal{L})$ occurs either at a vertex of $\mathcal{C}_k$ or a joint of the convex chain.*

Proof Assume for the sake of contradiction that the distance ratio is uniquely maximized for some **w** that is in the direction neither of a joint of the convex chain nor of a vertex of $\mathcal{C}_k$. But both $\mathcal{C}_k$ and the convex chain are piecewise linear, so must either be parallel or diverge in one direction. If they diverge, there is another $\mathbf{w}'$ in that direction with a larger distance ratio. If they are parallel, the nearest vertex or joint has the same distance ratio. So, **w** cannot be a unique maximum. □

The consequence of Lemma 4.3.4 is that it suffices to evaluate a given convex chain at its joints and the vertices of $\mathcal{C}_k$. Fig. 4.7 illustrates the dual space evaluation of some 2-regret minimizing sets. The area of positive regret, that which is under the lower envelope of the dual lines but above the contour, is shaded, and the error bars indicate the vertices at which the distance ratio is maximized.

To conclude the proof of Theorem 8 and tie the lemmata together, note that any lower envelope of a set of lines is, in fact, a convex chain and that, by convexity, any line can contribute at most one segment on the lower envelope. So, a convex chain with r line segments will have a corresponding size r set of lines, $\mathcal{L}'$. Because the convex chain minimizes the maximum distance ratio from $\mathcal{C}_k$ to O, it also minimizes the maximum k-regret ratio in primal space. So, this is the k-regret minimizing set of size r for $\mathcal{D}$. □

As an aside, if fewer than r points in $\mathcal{D}$ contribute segments to $\mathcal{C}_k$, then they are trivially a kRMS solution with zero k-regret. In such cases, computing vertices of $\mathcal{C}_k$ is sufficient to solve the problem (Corollary 4.3.5).

Corollary 4.3.5. *Let S be the set of line segments on $\mathcal{C}_k$. Then the set $R \subseteq \mathcal{D}$ that in dual space includes all of S is a k-regret minimizing set $R_r(\mathcal{D})$ if $|R| \leq r$.*

4.4 A Randomized Algorithm for General Dimension

Having shown the hardness of regret minimization in Section 4.2, we know that one cannot aspire towards a fast, optimal algorithm for kRMS in arbitrary dimension. However, we can still aim for a fast algorithm to find sets with *low* k-regret ratio; in this section, we describe a randomized, greedy kRMS algorithm that achieves this goal.

After first recalling the 1RMS algorithm of Nanongkai et al. [37], we extend it for 2RMS in Section 4.4.1. Then, we show how by introducing random partitioning with repeated trials, we can produce an effective kRMS algorithm for arbitrary k (Section 4.4.2).

Algorithm 7 is a simple yet effective greedy algorithm that expands an interim solution R point-by-point with the local optimum. For each interim solution R, Linear Program 1 below is run on every point $p \in \mathcal{D}$ to find the one that is responsible for the current maximum regret ratio. In the terminology of the previous section, each iteration from $1 < |R| \leq r$ finds the point on $\mathcal{C}_1$ farthest from R and adds that to the interim solution.

Algorithm 7 Greedy algorithm to compute 1RMS [37]

Input: $\mathcal{D}$; r
Output: $R \subseteq \mathcal{D}$, with $|R| = r$ and low 1-regret ratio
Let $R = \{p_i \in \mathcal{D}\}$, where p_i is a point in $\mathcal{D}$ with highest p_i^0.
while $|R| < r$ **do**
    Let q be first $p \in \mathcal{D}$.
    for all $p \in \mathcal{D} \setminus R$ **do**
        Let max_regret(p) be result of Linear Program 1 with input p, R.
        if max_regret(p)>max_regret(q) **then**
            Let q be p.
        end if
    end for
    Let $R = R \bigcup \{q\}$.
end while
RETURN R

Linear Program 1 below finds, given an interim solution $R \subseteq \mathcal{D}$ and a point $p \in \mathcal{D}$, the weight vector $\mathbf{w}$ that maximizes the 1-regret ratio of R relative to $R \bigcup \{p\}$. The 1-regret ratio is proportional to x, which is upper-bounded in constraint (4.2).

LINEAR PROGRAM 1.

$$\text{maximize } x \text{ s.t.} \tag{4.1}$$
$$\mathbf{p} \cdot \mathbf{w} - \mathbf{p}' \cdot \mathbf{w} \geq x \quad \forall \mathbf{p}' \in R \tag{4.2}$$
$$w_i \geq 0 \quad \forall 0 \geq i < d \tag{4.3}$$
$$\mathbf{p} \cdot \mathbf{w} = 1 \tag{4.4}$$
$$x \geq 0 \tag{4.5}$$

4.4.1 Extending 1RMS to 2RMS

To go from 1RMS to 2RMS, one needs to find points not top-ranked but 2-ranked and measure regret ratio with respect to them. Our 2RMS algorithm is largely unchanged from Algorithm 7 except for invoking Linear Program 2 instead. Linear Program 2 finds a weight vector **w** to maximize x, the regret ratio, and also determines with y in constraint (4.8) the amount by which the best point in $\mathcal{D} \setminus R \setminus \{p\}$ outscores p on **w**. So, if $y > 0$, then p is at best 2-ranked and an eligible candidate to add to R. On Line (8) of Algorithm 7, we specify an additional clause, that $y \geq 0$, to rule out 1-ranked points.

LINEAR PROGRAM 2.

$$\text{maximize } x - \varepsilon y \text{ s.t.} \tag{4.6}$$
$$\mathbf{p} \cdot \mathbf{w} - \mathbf{p}' \cdot \mathbf{w} \geq x \quad \forall \mathbf{p}' \in R \tag{4.7}$$
$$\mathbf{p}'' \cdot \mathbf{w} - \mathbf{p} \cdot \mathbf{w} \leq y \quad \forall \mathbf{p}'' \in \mathcal{D} \setminus R \setminus \{p\} \tag{4.8}$$
$$w_i \geq 0 \quad \forall 0 \leq i < d \tag{4.9}$$
$$\mathbf{p} \cdot \mathbf{w} = 1 \tag{4.10}$$
$$x \geq 0 \tag{4.11}$$
$$y \geq -\varepsilon \tag{4.12}$$

Constraint (4.8) is of an existential nature; so, there may be more than one point that outscores p in the direction of **w**, indicating that p is *not* 2-ranked. But if some other point p''' also outscores p on **w**, then either p'' or p''' will better maximize x than does p and be chosen instead. Note that we include y in the objective function to ensure it takes the

minimum valid value, making the 1-ranked case distinctive. The really small, positive real, ε, dampens the effect of y—we foremost want x to be maximized, even when it is paired with a large y. The value of ε should be chosen small enough that $x - \varepsilon y \approx x$.

4.4.2 Extending 2RMS to kRMS

Algorithm 8 Greedy algorithm to compute kRMS

Input: $\mathcal{D}$; r; k; T
Output: $R \subseteq \mathcal{D}$, with $|R| = r$ and low k-regret ratio
Let $R = \{p_i \in \mathcal{D}\}$, where p_i is point in $\mathcal{D}$ with highest p_i^0.
while $|R| < r$ **do**
  Let q be first $p \in \mathcal{D}$ and $\widetilde{\mathbf{w}} = \langle 0, \ldots, 0 \rangle$.
  for all $p \in \mathcal{D} \setminus R$ **do**
    for all i from 1 to T **do**
      Randomly partition $\mathcal{D} \setminus R \setminus \{p\}$ into $\mathcal{D}_0, \ldots, \mathcal{D}_{k-2}$
      Let max_regret(p) be result of Linear Program 3 with input $p, R, \mathcal{D}_0, \ldots, \mathcal{D}_{k-2}$.
      if max_regret(p) has all $x_i > 0$ **then**
        if max_regret(p)>max_regret(q) **then**
          Let q be p and $\widetilde{\mathbf{w}}$ be **w** from Linear Program 3.
        end if
        Break inner loop and go to next point.
      end if
    end for
  end for
  Let $S = \{q\}$ and **w**
  for all $p \in \mathcal{D} \setminus R$ **do**
    if $p \cdot \widetilde{\mathbf{w}} \geq q \cdot \widetilde{\mathbf{w}}$ **then**
      Let $S = S \bigcup \{p\}$.
    end if
  end for
  Let s be 'best' member of S with heuristic of choice.
  Let $R = R \bigcup \{s\}$.
end while
RETURN R

To solve the more general kRMS problem, we make use of Proposition 4.4.1 and randomness. The idea is that we decompose each iteration of the kRMS problem into a set of 2RMS problems and optimize for a common solution.

Proposition 4.4.1. *If $p = \mathcal{D}^{(k,\mathbf{w})}$, then there exists a partitioning of $\mathcal{D}$ into $\mathcal{D}_0, \ldots, \mathcal{D}_{k-2}$ such that $\forall \mathcal{D}_i, p = \mathcal{D}^{(2,\mathbf{w})}$.*

To rephrase Proposition 4.4.1, if p is k-ranked on $\mathcal{D}$ with respect to $\mathbf{w}$, then we can split $\mathcal{D}$ into $k-1$ partitions such that p will be 2-ranked on every one with respect to the same weight vector, $\mathbf{w}$. The key is that the partitions must each contain exactly one of the points higher ranked than p.

Without knowing apriori the weights of $\mathbf{w}$, it is challenging (and, we posit, an interesting open research direction) to construct such a partitioning. A random partitioning, however, may successfully separate the higher-ranked points into disjoint partitions and allow us to find $\mathbf{w}$ with Linear Program 3. Of course, a random partitioning may very well *not* produce such a separation, but then Linear Program 3 will report that p is not k-ranked and we can keep trying new random partitionings.

LINEAR PROGRAM 3.

$$\text{maximize } x - \varepsilon \sum x_i \text{ s.t.} \tag{4.13}$$

$$\mathbf{p} \cdot \mathbf{w} - \mathbf{p}' \cdot \mathbf{w} \geq x \qquad \forall \mathbf{p}' \in R \tag{4.14}$$

$$\mathbf{p}'' \cdot \mathbf{w} - \mathbf{p} \cdot \mathbf{w} \leq x_i \quad \forall \mathbf{p}'' \in \mathcal{D}_i, 0 \leq i \leq k-2 \tag{4.15}$$

$$w_i \geq 0 \qquad 0 \leq i < d \tag{4.16}$$

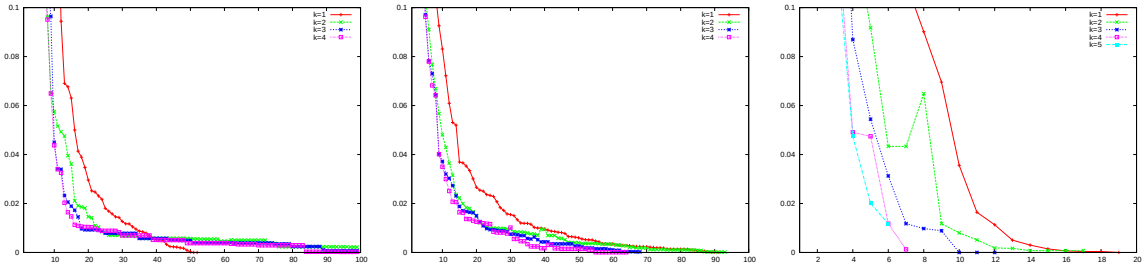
$$\mathbf{p} \cdot \mathbf{w} = 1 \tag{4.17}$$

$$x \geq 0 \tag{4.18}$$

$$x_i \geq -\varepsilon \qquad 0 \leq i \leq k-2 \tag{4.19}$$

So, we have Algorithm 8 to solve the kRMS problem. It is similar to Algorithm 7, except Linear Program 3 is executed several times for each point, each after randomly partitioning $\mathcal{D} \setminus R \setminus \{p\}$. If Linear Program 3 sets all $x_i > 0$, its solution is optimal; if it does not, either the partitioning was unlucky, R is still quite poor, or p cannot contribute to improving the interim R. So, we try another hopefully luckier partitioning until after a maximum number of trials that is dependent on k. There is a probability of $.1$ that 8 trials at $k = 3$ are all unlucky, for example. With Proposition 4.4.2, we can bound the number of partitioning trials with high probability; although, as will be evidenced in Section 4.5, 'unluckiness' is not necessarily so costly, anyway.

Proposition 4.4.2. *If one repeatedly partitions into m parts a dataset $\mathcal{D}$ with at least m points of interest, the probability of not obtaining a repetition in which each partition contains a point of interest after $\frac{3.16m^{2m}}{m!m!} - \frac{2.16m^m}{m!}$ trials is $\leq .1$.*



(a) Q1 - BB (b) Q1 - EN (c) Q1 - HH

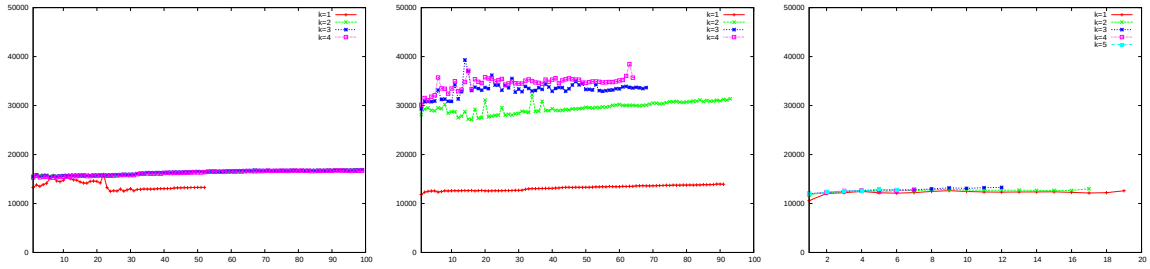
Figure 4.8: *Q1 plots: k-regret ratio.* A point on the plot shows the k-regret ratio (y-axis) achieved by a k-regret minimizing set of size $|R|$ (x-axis). Different values of k are represented by different curves. The range for the y-axis is $[0, .1]$ to make clearer the interesting range when subsets have passably low k-regret ratio.

PROOF SKETCH. The probability comes from the Chebyshev Inequality, given that the repeated partitioning is a Bernoulli Process with chance of success $\geq \frac{m!}{m^m}$.

Lastly, we call attention to Lines 18-24 of Algorithm 8. Once a maximal $\widetilde{\mathbf{w}}$ is discovered, there are k points in $p \in \mathcal{D}$ each for which $k\text{-regratio}(p, \widetilde{\mathbf{w}}) = 0$; so, any of these k points could be equally well chosen. We propose (and evaluate in the next section) three heuristics: KTH which simply uses the point q; MAX which uses the point with the highest score with respect to $\widetilde{\mathbf{w}}$; and MAG which uses the point p_i that maximizes $\sum_{j=0}^{d-1} p_i^j$. The purpose of these heuristics is to try to choose the point which best improves the performance of the algorithm on subsequent iterations, since they all perform equally well on the current one.

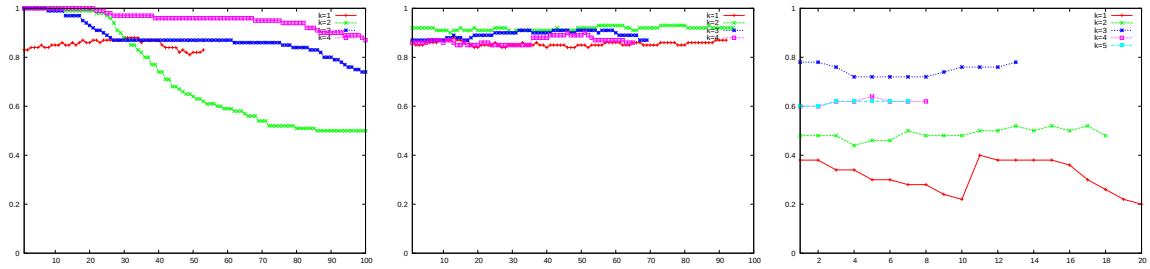
4.5 Experimental Evaluation

If you will recall, some primary contributions of this paper are introducing k-regret ratio, providing a general algorithm for computing a fixed-cardinality skyline approximation, and proving the hardness of kRMS for general k and d. In this section, we evaluate those first two contributions. Our overall goals are to compare the 1-regret ratio to the k-regret ratio for several values of k.



(a) Q3 - BB (b) Q3 - EN (c) Q3 - HH

Figure 4.9: *Q3 plots: execution time.* A comparison of wall time to run one instance of Linear Program 1 (in the case of 1RMS) or Linear Program 3 (in the case of kRMS, $k > 1$), averaged over 1000 trials. The x-axis gives $|R|$ and the y-axis gives time in microseconds.



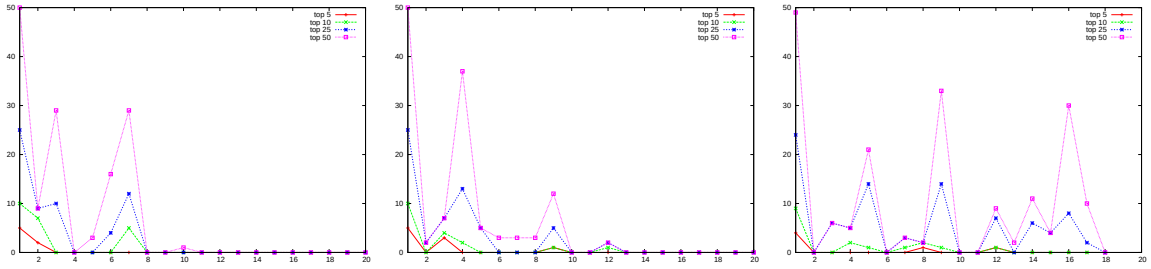
(a) Q4 - BB (b) Q4 - EN (c) Q4 - HH

Figure 4.10: *Q4 plots: score loss.* Every weight vector, if selected by the linear program, leads to heuristically choosing some maximal vector. These plots show how many weight vectors lead to a choice that scores within 2.5% of the k'th best score on the optimal maximal weight vector. For each size $|R|$ (the x-axis), the y-axis gives the percentage of tuples the incorrect selection of which would still produce a solution within the given bound of 'score loss.'

4.5.1 Setup

We implement the algorithms of Section 4.3.2 and 4.4 in C, using the MOSEK linear program (LP) solver.[1] We run the following experiments on a machine with an Intel Atom processor with two 800MHz cores and 1GB RAM, running Ubuntu 13.04. For the randomization, we run up to T trials, where T is selected such that the probability of failure is less than 0.01. We compute T in a separate, off-line calculation by repeatedly multiplying the single failure probability to the running product until the running product is less than 0.01, at which point T is set to the number of repetitions used.

[1] `http://www.mosek.com`



(a) Q5 - BB; $k = 2$ (b) Q5 - EN; $k = 3$ (c) Q5 - HH; $k = 4$

Figure 4.11: *Q5 plots: incidental correctness*. The number of points on which the application of the MAG heuristic to their maximal weight vector still results in selecting the point which maximizes the MAG heuristic on the optimal maximal weight vector. For each size $|R|$ (the x-axis), the y-axis gives the number of points that will *incidentally* produce the correct answer.

ID	Source	n	d	|Sky|
AI	`yvrdeals.com`	425	2	10
BB	`databasebasketball.com`	21961	5	200
EN	`kdd.ics.uci.edu`	178080	5	1183
HH	`usa.ipums.org/usa/`	862967	6	69

Table 4.3: Statistics for the four datasets used in our experiments.

4.5.2 Datasets

We use four datasets of varying size and dimension, summarized in Table 4.3. The all-inclusives (AI) dataset is the archetypal skyline dataset, trading off ratings and prices of hotels, that we used in the previous chapter. The basketball (BB) dataset contains points for each player/team/season combination for *rebounds*, *assists*, *blocks*, *fouls*, and *points scored*. The El Nino (EN) dataset consists of oceanographic readings like *surface temperature* and *wind speed*, taken at buoys placed in the Pacific Ocean. And the household (HH) dataset contains US census data for expenses such as *electricity* and *mortgage*. All attributes for all datasets have been normalized to the range [0,1] by subtracting the smallest value and then dividing by the range. Missing values have been replaced with the lowest value found in the dataset.

4.5.3 Experiment Descriptions

Broadly, we pose five questions that can be classified into three groups, expanded upon below.

Solution quality

Q1) *Regret ratio*. Our main comparison between 1-regret and k-regret is in calculating the k-regret ratio with respect to the number of tuples in R. In other words, at different values of k, how many points are required in order to guarantee users to be $x\%$ happy?

Q2) *Heuristic performances*. We proposed three heuristics, KTH, MAG, and MAX, to choose among the k points that all have k-regret-ratio of zero in the direction determined by the LP. While in Q1) we studied how quickly k-regret ratio is improved for different levels of k, here we fix k and contrast heuristics.

Efficiency

Q3) *Execution time*. We study the efficiency of the algorithm for several values of k by measuring wall time for the LP subroutine, which is the primary algorithmic difference for distinct values of k. This is a more meaningful metric than the overall wall time, because the difference between the cost of running the LP n times and of running the entire algorithm depends primarily on T, the number of partitioning trials, which is highly tunable.

Use of randomness

Q4) *Score loss*. Under a perfect partitioning, the LP will find a weight vector $\mathbf{w}$ with maximal k-regret ratio x. However, due to the use of randomness, it is is possible that T repeated 'unlucky' trials leads to missing this ideal solution, or even that mT repeated 'unlucky' trials leads to missing the m best LP solutions. We ask what percentage of the dataset must be missed by 'unlucky' randomness before losing 2.5% of the value of x.

Q5) *Incidental correctness*. With unlucky partitioning, the algorithm may miss the ideal greedy weight vector, $\mathbf{w}$, instead selecting $\mathbf{w}'$. Yet, it is possible still that the point maximizing the heuristic on $\mathbf{w}$ also maximizes the heuristic on $\mathbf{w}'$, creating an *incidental correctness*. We study the frequency with which this happens.

4.5.4 Discussion

Overall, our expectation was for a trade-off between quality and efficiency between $k = 1$ and $k > 1$. This is evident from the experiments. Interestingly, low values of $k > 1$ appear to hit a delicate balance between still being competitive in efficiency (because fewer randomized trials are required and the LP is still fast) and offering lower k-regret ratios on small subsets. Below we summarize the findings for each category of experiment.

Solution quality

On the AI dataset, for which we can compute optimal solutions, we find the following sets returned for manual comparison. At $k = 1$, $R = \{$(4.7,\$1367/pp),(3.4,\$847/pp),(4.6,\$1270/pp)$\}$; at $k = 2$, the third element is instead (3.9, \$1005/pp); and at $k \geq 3$, it requires only two resorts to satisfy every user. So, the sets at $k > 1$ appear more diverse at constrained sizes.

That observation is mirrored in the plots of Fig. 4.8, which show, as a function of $|R|$, the k-regret ratio achieved with our algorithm. On all datasets, there is a big jump from $k = 1$ to $k = 2$, more moderate jumps after that, and finally a stagnation around $k \geq 4$. The initial jumps are quite substantial. On the BB dataset, at $k = 3$ and $k = 4$, we achieve a k-regret ratio of $.02$ with a set R about half the size as is required for $k = 1$. The conclusion is that one can greatly improve the dataset approximation by increasing $k > 1$, but that small increases in k pay greater relative dividends.

We omit plots for Q2 because the are uninteresting, showing that MAG outperforms the other two heuristics quite consistently. The exception is for really high $|R|$ on the basketball dataset, where MAG struggles to reach a k-regret ratio of 0, but it is subsets of size, say, less than thirty that are interesting, anyway. The success of MAG partly explains the improvement with $k > 1$ vs. $k = 1$. At $k = 1$ there is one and only one point that achieves 0 k-regret ratio in the direction of $\mathbf{w}$. At $k > 1$, on the other hand, there are k points from which to maximize the MAG heuristic.

Efficiency

The plots in Fig. 4.9 show, as a function of $|R|$, the average execution time for a single run of the pertinent linear program (LP). One can interpret this as the chronological evolution of the LP running time, for each outer loop of the algorithm. The results are very data-dependent, both in the shapes of the curves and the size of the gap from the $k = 1$ curve to the $k > 1$ curves. The interesting conclusion here is that the addition of n constraints and $k - 1$ variables does not render the algorithm uncompetitive—instead it remains comparable. This is especially interesting at $k = 2$, where no randomized trials are necessary.

Use of randomness

The results on randomness were especially interesting and suggest interesting future research directions. The plots in Fig. 4.10 show, as a function of $|R|$, what percentage of the top 100 sub-optimal points produce a score close to optimal—characterizing the cost of selecting a sub-optimal point. The percentages are quite high. For example, at $k = 3$,

on all datasets and all values $|R|$, over 70% of the top 100 points produce a score within 2.5% of optimal. Therefore, there is a very strong resilience to 'bad luck': $70T$ trials must all fail in order to lose 2.5%. It is an interesting question to what extent this can be guaranteed theoretically, especially since the relative order for different values of k changed on different datasets. The conclusion that we draw is that one can confidently lower T (and thus decrease running time) without incurring much of a hit to the quality of the solution.

Secondly, the plots in Fig. 4.11 show, as a function of $|R|$ the number of the first $5, 10, 25,$ and 50 points that lead to incidental selection of the best choice. This happens surprisingly often, considering that one might not expect it at all, for $|R| \leq 10$ and almost without fail for $|R| = 1$. This suggests that it could be possible to decrease T for certain small values of $|R|$, where one is likely to obtain the correct value even in the case of unlucky partitioning.

Summary

We briefly summarize the results by value of k. For $k = 1$, the LP is consistently faster, but not always by much. The efficiency comes at a cost of being able to achieve a low 1-regret ratio with subsets of size less than 25 on the two more challenging datasets. At $k = 2$, the LP is nearly as fast and run no more often, and the 2-regret ratio drops substantially faster. At $k = 3$, the 3-regret ratio drops faster yet, but now at the cost of running T trials per data point. However, T can perhaps be held quite low without losing much score, and even lower yet for $|R| \leq 10$ under the anticipation of some incidental correctness. Finally, for $k \geq 4$, the k-regret ratio drops not much faster than at $k = 3$ and T goes up quite fast.

4.6 Bibliographic Notes

The idea to represent an entire dataset by a few representative points for multi-criteria decision making has drawn much attention in the past decade, since the introduce of the Skyline operator by Börzsönyi et al. [6]. However, the susceptibility of the skyline operator to the curse of dimensionality is well-known. Chan et al. [8] made a compelling case for this, demonstrating that on the NBA basketball dataset (as it was at the time), more than 1 in 20 tuples appear in the skyline in high dimensions. Consequently, there have been numerous efforts to derive a representative subset of smaller size (e.g., [7, 27, 54, 56]), especially one that presents very distinct tuples (e.g., [17, 43]) or has a fixed size (e.g., [28, 29, 45]).

The work here on k-regret minimizing sets is novel, not yet published, although under review for submission at a top-tier conference [11]. The less general 1-regret minimizing sets are relatively new in the lineage of these efforts. When introduced by Nanongkai et al. [37], the emphasis was on proving that the maximum regret ratio is bounded by:

$$\frac{d-1}{(r-d+1)^{d-1}+d-1}.$$

Naturally, this bound holds for the generalisation introduced in this paper, since $k\text{-regratio}(R, \mathbf{w}) \leq (k-1)\text{-regratio}(R, \mathbf{w})$. As far as we know, this work is the first to address computational questions around k-regret minimizing sets, certainly for $k > 1$.

Regret minimizing sets presuppose that linear top-k queries are of interest, a class of queries that has been well studied and has been surveyed quite thoroughly by Ilyas et al. [24]. The use here of duality is fairly common (e.g., [15,26,44]) as is the emphasis on (layers of) lower envelopes (e.g., [9,57]). Transforming points into dual space in two dimensions often leads to the employment of plane sweep algorithms [18] and the availability of many results on arrangements of lines. For example, Agarwal et al. [2] give bounds on the the number of edges and vertices that can exist in a chain (such as $\mathcal{C}_k$) through an arrangement. The top-k rank contours of Chester et al. [12], which were proposed to answer monochromatic reverse top-k queries [47] and are central to our two dimensional algorithm, fit into this category. It is an interesting question whether duality can help in higher dimensions and also whether there exists a strong connection between reverse top-k queries and k-regret minimization as the application of these results may imply.

Lastly, *anytime* skyline algorithms can be halted mid-execution and output a non-optimal solution [31]. Regret minimizing sets are well suited to these interactive scenarios [36]; so, it is reasonable to believe that k-regret minimizing sets may also be suitable.

4.7 Final Remarks

The 1-regret minimizing set is a nice alternative to the skyline as a succinct representation of a dataset, but suffers from rigidly fitting the top-1 for every query. We generalised the concept to that of the *k-regret minimizing set*, which represents a dataset not by how closely it approximates every users' top-1 choice, but their top-k choice. Doing so permits simultaneously achieving a lower k-regret ratio while making the representative subsets much smaller.

For the special case of $d = 2$, we give an efficient, exact algorithm based on the dual

space insight that the k-regret minimizing set corresponds to the convex chain closest to the top-k rank contour. The algorithm uses dynamic programming and plane sweep to navigate the space of convex chains. For general dimension, we first resolve a conjecture that computing a 1-regret minimizing set is NP-Hard and extend the hardness result to k-regret minimizing sets. Then, we give a randomized, greedy algorithm based on linear programming to find a subset with *low* k-regret ratio. In comparison to computing subsets with low 1-regret ratio, we show that its execution time is very competitive and its results are much stronger. Furthermore, the randomization component of the algorithm opens several very interesting research directions.

In the broader context of this dissertation's objective, the k-regret minimizing set is a leading-edge concept for summarizing large datasets that will be subject to preference querying. We believe it will find its way into industrial systems for helping users to navigate large and complex datasets to find the tuples most relevant to them.

Chapter 5

Future Directions

An exciting aspect of research is that it always produces more questions than answers. Here follows a number of research directions opened by the results in this dissertation that could lead to substantial advancement of the state-of-the-art.

Trade-off between computability and succinctness of summary operators

We showed in Theorem 7 that kRMS, even 1RMS, is hard. Yet, it is known that computing the skyline of a dataset can be done in polynomial time. On the other hand, kRMS provides a much more succinct representation of the dataset than the skyline does. In addition to this, the complexity of computing k-contours is unknown in general, and it would seem to be the minimally succinct summarization operator that does not require approximation of $\mathcal{D}$. Clearly, there is a trade-off between succinctness and efficient computability for these different summary operators. There is a broad spectrum of possible definitions of summary operators and exploring that space to find a 'sweet spot' that balances the membership in P that the skyline offers with the succinctness of kRMS is an interesting endeavour. Similarly, can it be said that all size-r constrained summary operators are destined to be hard?

Theoretic investigation into partitioning characteristics of datasets as they relate to (reverse) top-k queries

Chapter 4 introduced a very novel approach to the reverse top-k problem, decomposing it into $k-1$ reverse top-2 problems and looking for a common solution. The success of this technique in that chapter depended on randomness, which is an excellent technique,

but requires repeating the same computation in order to increase the probability of success. The empirical study in Section 4.5 opened up some very intriguing and connected research questions. Can one design a higher-probability partitioning than just random balls in buckets? And from a number-theoretic perspective, what can be said about the relation between best, second-best, &c., weight vectors for maximizing the LP. More broadly stated, given the successful use of randomization in that algorithm, what more can we say about applying it to problems of this nature?

Monochromatic reverse top-k queries in higher dimensions

In Chapter 3, we give a polynomial algorithm for two dimensions that uses techniques (viz., plane sweep) which are known not to generalize to higher dimension. Interestingly, previous work also focused on the two dimensional case. But what can be said about arbitrary dimension? A first obvious direction of research is in resolving the hardness of resolving mRTOP queries and of constructing k-contours in arbitrary dimension. But a particularly interesting question is how much easier and faster is it to compute the set of tuples contributing to the k contour than to compute its vertices, if at all? Because if the tasks are of equal difficulty, it seems one might be able to straight-forwardly apply Linear Program 3 to these problems.

Applying our techniques to other new and strongly related problems

Preference queries, even just linear top-k queries, have attracted a lot of research attention in the past decade. Among that research has been the definition of copious new, similarly-inspired problems. Many of the contributions in this paper were the result of applying new techniques, so it is a curious question how widely those techniques can be applied to the diverse set of other top-k-related problems. For example, *Why not?* queries [19] are quite similar to reverse top-k queries, in that they pose the question of how to modify a query to include a given tuple. In another direction, some especially practical research efforts have looked at combining top-k queries with other standard database operators, such as *group by* [30] or *join* [22, 23, 49]. How well do the techniques in this work suffice when the top-k query is combined with another operator?

Algorithmic improvements for the kRMS problem and $\mathcal{C}_k$ construction

Our algorithm in the general case for kRMS is heuristic; so, it is reasonable to expect that one may be able to improve the algorithm. So, a first natural question is whether one can provide any theoretical guarantees for the LP-approach proposed both in Chapter 4 and by Nanongkai et al. [37]. Alternatively, are there any algorithmic means of finding either solutions closer to optimal or techniques that are even faster in practice (perhaps at the expense of accuracy)? Similarly, while our construction algorithm for $\mathcal{C}_k$ is not heuristic, it is the first effort in literature. So, it is worth investigating whether the construction cost can be reduced from $\mathcal{O}(n^2)$.

Interplay between view-based top-k resolution and convex chain 'joints'

Theorem 8 showed that the dual space equivalent in two dimensions of kRMS was finding a convex chain near the k'th contour. An interesting corollary to the theorem is that there are finitely few especially salient user weight vectors: the vertices of the contour and the joints of the convex chain. In fact, this holds in higher dimensions, too, where convex chains would theoretically generalize to convex polytopes and the analogue to the 'joints' would be intersection points of (hyper-)planes. On another note, view-based top-k query resolution [16, 40, 52] pre-materializes views with solutions to anticipated user queries. There is an inherent connection between these problems in that the 'joints' are exactly the set of interesting user queries and a materialized view for each (which may be prohibitive) would exactly service all users. Furthermore, k-regret ratio may provide a means of assessing the impact of each materialized view. Exploring the inter-connections between these concepts and problems could lead to fascinating and highly practical results.

Bibliography

[1] Pankaj K. Agarwal, Lars Arge, Jeff Erickson, Paulo G. Franciosa, and Jeffrey Scott Vitter, *Efficient searching with linear constraints*, Journal of Computer and System Sciences **61** (2000), no. 2, 194–216.

[2] Pankaj K. Agarwal, Boris Aronov, and Micha Sharir, *On levels in arrangements of lines, segments, planes, and triangles*, Proc. Symposium on Computational Geometry (New York, NY, USA), ACM, 1997, pp. 30–38.

[3] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi, *The priority r-tree: A practically efficient and worst-case optimal r-tree*, Proc. SIGMOD '04, 2004, pp. 347–358.

[4] Lars Arge and Jeffrey Scott Vitter, *Optimal external memory interval management*, SIAM Journal of Computing **32** (2003), no. 6, 1488–1508.

[5] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars, *Computational geometry: Algorithms and applications*, 3rd ed., Springer-Verlag TELOS, Santa Clara, CA, USA, 2008.

[6] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker, *The skyline operator*, Proc. International Conference on Data Engineering (ICDE) (Washington, DC, USA), IEEE Computer Society, 2001, pp. 421–430.

[7] Chee-Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang, *Finding k-dominant skylines in high dimensional space*, Proc. ACM Special Interest Group on Management of Data (SIGMOD) (New York, NY, USA), ACM, 2006, pp. 503–514.

[8] ———, *On high dimensional skylines*, Proc. International Conference on Extending Database Technology (EDBT) (Berlin, Heidelberg), Springer-Verlag, 2006, pp. 478–495.

[9] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith, *The onion technique: indexing for linear optimization queries*, Proc. ACM Special Interest Group on Management of Data (SIGMOD) (New York, NY, USA), ACM, 2000, pp. 391–402.

[10] Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides, *Indexing for vector projections*, Proc. Database Systems for Advanced Applications (DASFAA), Springer-Verlag, April 2011, pp. 367–376.

[11] ———, *Computing k-regret minimizing sets*, arXiv:1207.6329 [cs.DB], 2013, Under review for VLDB 2014.

[12] ———, *Indexing reverse top-k queries in two dimensions*, Proc. Database Systems for Advanced Applications (DASFAA), Springer Berlin Heidelberg, April 2013, pp. 201–208.

[13] ———, *Monochromatic reverse top-k queries in two dimensions*, The Computer Journal (2013), Under revision.

[14] ———, *Threshold projection queries*, Information Systems (2013), Under revision.

[15] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Nikos Sarkas, *Ad-hoc top-k query answering for data streams*, Proc. Very Large Databases (PVLDB), VLDB Endowment, 2007, pp. 183–194.

[16] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis, *Answering top-k queries using views*, Proc. Very Large Databases (PVLDB), 2006, pp. 451–462.

[17] Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, Richard J. Lipton, and Jim Xu, *Representative skylines using threshold-based preference distributions*, Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (Washington, DC, USA), ICDE '11, IEEE Computer Society, 2011, pp. 387–398.

[18] Herbert Edelsbrunner and Lionidas J. Guibas, *Topologically sweeping an arrangement*, Proc. 18^{th} ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 1986, pp. 389–403.

[19] Zhian He and Eric Lo, *Answering why-not questions on top-k queries*, Transactions on Knowledge and Data Engineering (TKDE) (2012), preprint.

[20] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou, *Prefer: a system for the efficient execution of multi-parametric ranked queries*, Proceedings of the 2001 ACM SIGMOD international conference on Management of data (New York, NY, USA), SIGMOD '01, ACM, 2001, pp. 259–270.

[21] John Hugg, Eynat Rafalin, Kathryn Seyboth, and Diane Souvaine, *An experimental study of old and new depth measures*, In Proc. Workshop on Algorithm Engineering and Experiments (ALENEX06), Lecture Notes in Computer Science, Springer, 2006, pp. 51–64.

[22] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid, *Supporting top-k join queries in relational databases*, Proc. Very Large Databases (PVLDB), 2003, pp. 754–765.

[23] ———, *Supporting top-k* join *queries in relational databases*, VLDB Journal **13** (2004), no. 3, 207–221.

[24] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman, *A survey of top-k query processing techniques in relational database systems*, ACM Compututing Surveys **40** (2008), no. 4, 11:1–11:58.

[25] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, 1972, pp. 85–103.

[26] Jongwuk Lee, Hyunsouk Cho, and Seung-Won Hwang, *Efficient dual-resolution layer indexing for top-k queries*, Proc. ICDE, IEEE Computer Society, 2012, pp. 1084–1095.

[27] Jongwuk Lee, Gae-won You, and Seung-won Hwang, *Personalized top-k skyline queries in high-dimensional space*, Information Systems **34** (2009), no. 1, 45–61.

[28] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang, *Select stars: the k most representative skyline operator*, Proc. International Conference on Data Engineering (ICDE), 2007, pp. 86–95.

[29] Hua Lu, Christian S. Jensen, and Zhenjie Zhang, *Flexible and efficient resolution of skyline query size constraints*, Transactions on Knowledge and Data Engineering (TKDE) **23** (2011), no. 7, 991–1005.

[30] Matteo Magnani and Ira Assent, *From stars to galaxies: skyline queries on aggregate data*, Proc. International Conference on Extending Database Technology (EDBT), 2013, pp. 477–488.

[31] Matteo Magnani, Ira Assent, and Michael Lind Mortensen, *Anytime skyline query processing for interactive systems*, Proc. DBRank, 2012.

[32] Jiří Matoušek, *Reporting points in halfspaces*, Computational Geometry: Theory and Applications **2** (1992), no. 3, 169–186.

[33] ———, *Geometric range searching*, ACM Computing Surveys **26** (1994), no. 4, 422–461.

[34] Jiří Matoušek and Otfried Schwarzkopf, *Linear optimization queries*, Proceedings of the 8th Annual Symposium on Computational Geometry, ACM, June 1992, pp. 16–25.

[35] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu, *Reverse data management*, Proc. VLDB Endowment **4** (2011), no. 12, 1490–1493.

[36] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino, *Interactive regret minimization*, Proc. ACM Special Interest Group on Management of Data (SIGMOD), 2012, pp. 109–120.

[37] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun Xu, *Regret-minimizing representative databases*, Proc. Very Large Databases (PVLDB) **3** (2010), no. 1, 1114–1124.

[38] Sayan Ranu and Ambuj K. Singh, *Answering top-k queries over a mixture of attractive and repulsive dimensions*, Proc. VLDB Endowment **5** (2011), no. 3, 169–180.

[39] Peter J. Rousseeuw and Mia Hubert, *Depth in an arrangement of hyperplanes*, 1999.

[40] Norvald H. Ryeng, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg, *Efficient distributed top-k query processing with caching*, Proc. Database Systems for Advanced Applications (DASFAA), 2011, pp. 280–295.

[41] Micha Sharir, *Arrangements in higher dimensions: Voronoi diagrams, motion planning, and other applications*, In Proc. 4th Workshop Algorithms Data Struct, Springer-Verlag, 1995, pp. 109–121.

[42] Alexander S. Szalay, Jim Gray, George Fekete, Peter Z. Kunszt, Peter Kukol, and Ani Thakar, *Indexing the sphere with the hierarchical triangular mesh*, CoRR **abs/cs/0701164** (2007).

[43] Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei, *Distance-based representative skyline*, Proc. International Conference on Data Engineering (ICDE) (Washington, DC, USA), IEEE Computer Society, 2009, pp. 892–903.

[44] Panayiotis Tsaparas, Nick Koudas, and Themistoklis Palpanas, *Ranked join indices*, Proc. International Conference on Data Engineering (ICDE), 2003, pp. 277–288.

[45] Akrivi Vlachou, Christos Doulkeridis, and Maria Halkidi, *Discovering representative skyline points over distributed data*, Proc. Int. Conf. on Sci. and Statist. Database Manag. (SSDBM), 2012, pp. 141–158.

[46] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg, *Reverse top-k queries*, Proc. International Conference on Data Engineering (ICDE), IEEE, March 2010, pp. 365–376.

[47] ———, *Monochromatic and bichromatic reverse top-k queries*, Transactions on Knowledge and Data Engineering (TKDE) **23** (2011), no. 8, 1215–1229.

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

[49] Akrivi Vlachou, Christos Doulkeridis, and Neoklis Polyzotis, *Skyline query processing over joins*, Proc. ACM Special Interest Group on Management of Data (SIGMOD), 2011, pp. 73–84.

[50] Biyan Wang, Zhengqing Dai, Cuiping Li, and Hong Chen, *Efficient computation of monochromatic reverse top-k queries*, Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on (Los Alamitos, CA, USA), vol. 4, IEEE Computer Society, August 2010, pp. 1788–1792.

[51] E. J. W. Whittaker, *The stereographic projection*, no. 11, University College Cardiff Press, Cardiff, Wales, 1984.

[52] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood, *Efficient top-k query answering using cached views*, Proc. International Conference on Extending Database Technology (EDBT), 2013, pp. 489–500.

[53] Dong Xin, Chen Chen, and Jiawei Han, *Towards robust indexing for ranked queries*, Proceedings of the 32nd international conference on Very large data bases, VLDB '06, VLDB Endowment, 2006, pp. 235–246.

[54] Man Lung Yiu and Nikos Mamoulis, *Multi-dimensional top-k dominating queries*, VLDB Journal **18** (2009), no. 3, 695–718.

[55] Albert Yu, Pankaj K. Agarwal, and Jun Yang, *Processing a large number of continuous preference top-k queries*, Proc. ACM SIG Management of Data (New York, NY, USA), ACM, 2012, pp. 397–408.

[56] Zhenjie Zhang, Xinyu Guo, Hua Lu, Anthony K.H. Tung, and Nan Wang, *Discovering strong skyline points in high dimensional spaces*, Proc. Int. Conf. on Information and Knowledge Manag. (CIKM), 2005, pp. 247–248.

[57] Lei Zou and Lei Chen, *Pareto-based dominant graph: An efficient indexing structure to answer top-k queries*, IEEE Trans. Knowl. Data Eng. **23** (2011), no. 5, 727–741.

[58] Yijun Zuo and Robert Serfling, *Structural properties and convergence results for contours of sample statistical depth functions*, Annals of Statistics **28** (2000), no. 2, 483–499.