RANKED SELECTION INDEXES FOR LINEAR PREFERENCE QUERIES

A Thesis by

Sanjaya Singh

Bachelor of Computer Engineering, Pokhara University, Nepal, 2005

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of Graduate School of
Wichita State University
in the partial fulfillment of
the requirements for the degree of
Master of Science

December 2011

RANKED SELECTION INDEXES FOR LINEAR PREFERENCE QUERIES

The following faculty have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Science.

_____

Prakash Ramanan, Committee Chair

_____

Achita (Mi) Muthitacharoen, Committee Member

_____

Abu Asaduzzaman, Committee Member

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude to my supervisor Dr. Prakash Ramanan for his patience, personal guidance and counsel. Without him, the initiation and the completion of the project would be out of question. His understanding nature, encouragement, constructive comments and good editing skills have provided a good basis for the present thesis. I could not have asked for a better professor and a supervisor to complete my graduate studies.

In addition, I am also grateful to Rita Subba, Samir Sharma and Srijana Pokharel for their strong support and gentle push towards the completion of my thesis. They made it their business to make sure that I stayed motivated and focused. I am forever grateful for their comforting presence in my life.

I also wish to thank all my graduate friends for sharing ideas and providing invaluable assistance during the completion of my thesis.

Lastly, I am thankful to my family for their faith in me and for loving me who I am.

ABSTRACT

Data entities from various data sources could be ordered according to a variety of attributes associated with those entities. These orderings result in a ranking of entities in terms of the values in the attribute domains. In query processing, user preferences are desired to be tied to values of specific rank attributes. A way to incorporate such preference is by utilizing a function f that combines user preferences and rank attribute values and returns numerical value. Top-k queries seek to identify the tuples with the highest numerical value.

We consider the top-k selection query on relational database:

SELECT * FROM S ORDER BY $f(t)$ LIMIT k

We propose efficient indexes on S to find the top-k tuples, for a given linear monotone preference function f. The efficiency of our approach depends on the number of *dimensions*: the number of rank attributes used to compute f. We present efficient algorithms for two dimensions. Our results for two dimensions improve upon Tsaparas et. al. Our approach is based on *convex layers*, which is more appropriate for linear preference function.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

Data entities from various data sources could be ordered according to a variety of attributes associated with those entities. These orderings result in a ranking of entities in terms of the values in the attribute domains. These attributes associated with the entities can be of numerical or categorical values. In query processing, user preferences are desired to be tied to values of specific attributes.

Such values could reflect different quantities of interest for these data entities, such as physical characteristics, qualities, reliabilities etc. For example, consider a database of hotels ordered by the *ratings* (star classifications) or *prices* of rooms; lists of vehicles ranked according to *brand*, *color*, *model*, *mileage*, *price* or lists of vehicle suppliers ranked by their *credibility* or *service quality* (obtained by recording user experience with them over time). The attributes which have numerical values with real numbers (non-negative) like *price* of room for aforementioned hotel database or *mileage* (miles per gallon) and *price* of a vehicle in the “vehicle” database are considered as rank attributes which are used in a preference function. Whereas, other attributes like *brand*, *color*, *model* in vehicle database or *rating* of the hotels in hotel database are considered as non-rank attributes. These non-rank attributes are generally categorical in nature. The values for non-rank attributes can have numerical values. Like, for star *ratings* from one to five. Also, for vehicle suppliers, *credibility* can be rated from bad to excellent or

could have numerical values ranging from one to five respectively. But, these values are not considered for rank attributes. In order for attributes to be considered as rank attribute, it should have real non negative numerical values. Hence, the domain of these rank attributes depends on their semantics.

The presence of these rank attributes alongside data entities leads to better query processing capabilities and functionalities. Of particular importance is the ability of answering queries with the goal of optimizing functions that relate user preferences to rank attribute values. For example, consider a relation of vehicles, where there are different attributes: *brand*, *color*, *model*, *vehicle type*, *mileage* (miles per gallon), *reliability*, *price*; out of these, data entities (cars) can be ranked based on various attributes of cars like *mileage* (miles per gallon) and *reliability*. We do not consider *price* to be rank attribute, as higher the price, lesser value the user will assign to it. We need to consider higher value of rank attribute as more desirable. Users specify their preferences to these specific attributes. Usually, preferences are represented in terms of numerical weights assigned by users, to the rank attributes. Query processors include functionality that utilize these attribute weights to derive values for individual entities. Various techniques have been developed in order to answer queries with the goal to identify results that optimize these functions. An exemplary instance is a query that sought to quickly identify data entities that yield best values for rank among all entities in the database. At an abstract level such queries can be considered as generalized forms of selection queries.

TABLE 1

RANK ATTRIBUTES (*MILEAGE* AND *RELIABILITY*) FOR VEHICLE RELATION

VID	**BRAND**	**COLOR**	**VEHICLE_TYPE**	**PRICE**	**MILEAGE (MPG)**	**RELIABILITY(out of 10)**
V35	Honda	Black	SUV	22k	25	8
V36	Honda	Red	Sedan	18k	30	7
V38	Toyota	Silver	Coupe	21k	35	6
V40	Ford	Blue	Truck	19k	20	4

Here, vehicles consist of seven attributes *vehicle id*, *brand*, *color*, *vehicle type*, *price, mileage*, and *reliability*. Of them, *mileage* and *reliability* are rank attributes, while *brand*, *color, vehicle type* and *price are* non-rank attributes. Attributes *mileage* and *reliability* determine the mileage and reliability of a vehicle containing real numerical values from which user can determine the score of the vehicle.

A user (buyer) interested in purchasing a vehicle will depend on these two rank attributes to which he/she will assign some numerical weights. Rank attributes will provide greater flexibility in query specification. It is crucial to capture user interest/preferences towards rank attributes and efficiently support such queries. User preferences for rank attributes are obtained by allowing users to specify numerical

weights for those rank attributes. Larger the weight greater the preference/interest of the user towards that rank attribute.

Assuming the existence of functions that tie user preferences to rank attribute values and return numerical value, top-k queries seek to identify the tuples with highest scores.

In this paper, we consider the top-k selection query: Output the top-k tuples, from a relational table. The most general form of such a query is as follows:

SELECT $*$ FROM S ORDER BY $f(t)$ LIMIT k

This query outputs the top-k tuples from the relation S, according to f and limited by k without the use of filter or WHERE clause. f is a preference function that assigns a non-negative score to each tuple t in S; $f(t)$ is based on some rank attributes of t. Although this query seems very restrictive, it is an important special case that has received much attention. [16, 18, 25, 33, 34, 52, 53, 57, 73, 74, 75] deal with this kind of query with minor variations.

Here, we consider efficient indices, as the solution for this problem. This solution consists of two steps:

1) Index construction: The relation S is known at index construction time. f is not known but we restrict f to the important class of linear monotone functions. This index would work on all f in this class.

2) Query processing: Given a preference function f, use the index to quickly output the top-k tuples.

Our method uses techniques from computational geometry Preparata and Shamos [62], to efficiently carry out both these steps. We present an efficient algorithm for the following special case: $f(t)$ is based on two rank attributes and we want the top-2 results (i.e. k=2).

Most of the previous works are based on the maximal layers of S. As shown in Chang et al. [16], for linear monotone preference a function, using convex layers of S is more appropriate. Our index is based on convex layers.

Ilyas et al. [37] classifies previous works on top-k queries based on five different design dimensions. On the first dimension (query model), we start with top-k *selection*. Second dimension, data model is certain data; third dimension, implementation, is at the application level, and fourth dimension, ranking function, is linear monotone. The remaining dimension, fifth, is the method of accessing data through sorting technique or random access based on partial preference values (scores); this is not directly relevant to our approach.

In chapter 2, we discuss related works. In chapter 3, we present our notations and definitions. In chapter 4, we define the maximal and convex layers that are relevant to our top-k queries. In chapter 5, we propose an algorithm for construction of our index in two dimensions. In chapter 6, we present our conclusion and directions for future research.

CHAPTER 2

RELATED WORKS

There are lots of work related to top-k query and skyline in ranked joined index.

Bartolini et al. [4] proposed Efficient Sort-Based Skyline Evaluation which introduces SaLSa algorithm that exploits the idea of presorting the input data so as to effectively limit the number of tuples to be read and compared. In this technique they consider symmetric sorting functions and criterion to improve the performance, given the data distribution.

Bruno et al. [9] proposed the translation of top-k query into a single range query for RDBMS to process efficiently. It uses a range query to evaluate a top-k query by exploiting the statistics available to an RDBMS. For this, it uses database histograms to map a top-k query to a suitable multi-attribute range query such that k closest matches are likely to be included in the answer to the generated range query.

Chang and Hwang [15] proposed to overcome the shortcoming of previous standard sort-merge framework for ranked queries to produce top-k answers. They developed the formal principle of “necessary probes,” which determines if a probe is absolutely required. They proposed an algorithm MPro which is optimal with minimal probe cost; their algorithm scales well with easy parallelization.

Chen and Lian [20] proposed metric skyline, whose dynamic attributes are defined in the metric space. Here, they retrieve skyline points with dynamic attributes in the metric space. The sets of data objects and query objects are the input for metric skyline query: Those data objects whose attribute vectors are not dominated by other data object, where the attribute vector of a data object is defined as an n-dimensional vector consisting of metric distances from this object to n query points.

Hwang and Chang [35] takes a cost-based optimization approach by runtime search over a space of algorithms, cost-based optimization is general across a wide range of access scenarios, yet adaptive to the specific access costs at runtime. It defines the logical tasks of top-k queries as building blocks to identify a comprehensive and focused space for top-k queries. It has efficient search schemes over such a space for identifying the optimal algorithm.

Ilyas et al. [36] introduced a rank-join algorithm that makes use of the individual orders of its inputs to produce join results ordered on a user-specified scoring function. It proposes to rank the join results progressively during the join operation. It introduces two physical query operators based on variants of ripple join that implement the rank-join algorithm. The operators are non-blocking and can be integrated into pipelined execution plans.

Chan et al. [12] proposed finding meaningful Skylines in multi-dimensional Space. For this, they propose k-dominant skyline which relaxes the idea of dominance to k-dominance. A point p k-dominates another point q if there are k dimensions in which p

is better than or equal to q and is better in at least one of these k dimensions. Further, it says a point that is not k-dominated by any other point is in the k-dominant skyline.

To find the top-k results from join queries, Tsaparas et al. [73] proposed a technique called ranked join index. This depends on user preference and scoring function. For finding top-k, the approach consist of two steps: index construction and query processing.

Our proposed work is closely related to Tsaparas et al. [73] for finding top-k tuples in two-dimensional space. In Tsaparas et al. [73], ranked join index is created which reprocesses dominating set and constructs index on its elements. In the proposed algorithm, a vector is let to sweep the 2D plane. Each time it crosses a separating vector, the order of dominating set is changed by swapping two adjacent tuples. Hence, generating the top-k tuple in dominating sets. To find out the top-k selection in skyline we will be using convex hull algorithm studied in computational geometry Preparata and Shamos [62]. The output of top-k will depend on the preference function. Convex and maximal layers are used to determine the dominating sets. The number of polygonal regions while creating top-k selection index is same as that of Tsaparas et al. [73], as is the space required to store the index. But our algorithm excels in time required to create the index.

CHAPTER 3

PROBLEM DEFINITION

Let R (resp. R_+) denote the set of all (resp. non-negative) real numbers. Consider a relation $S(A_1, A_2, \ldots\ldots, A_d, B_1, B_2, \ldots\ldots\ldots, B_m)$.

A_i $(1 \leq i \leq d)$ are rank attributes whose domains are subsets of R_+. B_j $(1 \leq j \leq m)$ are non-rank attributes with arbitrary domains; these attributes may be numerical or categorical. For a tuple $t \in S$. Let $A_i(t)$ (resp. $B_j(t)$) denote the value of attribute A_i (resp. B_j) of t. Let n denote the number of tuples in S.

Definition 3.1.

[Preference function] A preference function is a function $f : R_+^d \rightarrow R_+$. For each tuple $t \in S$, it assigns a non-negative score $f(t)$, based only on its rank attribute values $A_i(t)$. ◊

TABLE 2

NOTATIONS USED IN CHAPTER 3

ITEMS	DESCRIPTIONS
R_+	set of all non-negative real numbers
$S(A_1, A_2, \dots, A_d, B_1, B_2, \dots, B_m)$	relation
$A_i\ (1 \le i \le d)$	rank attributes; domains are subsets of R_+
$B_j (1 \le j \le m)$	non-rank attributes; numerical or categorical
$A_i(t)\ or\ B_j(t)$	value of attribute $A_i\ or\ B_j\ of\ t \in S$
n	number of tuples in S
$f: R_+^d \to R_+$	preference function; for each $t \in S$, assigns a score $f(t)$, based only on the values of its rank attributes
$t \preccurlyeq t'$	t is dominated by t'; based only on values of rank attributes

TABLE 2 (contd.)

$t \prec t'$	t is strictly dominated (sdominated) by t'
t, t' incomparable	$t \preccurlyeq t' and\ t' \preccurlyeq t$
TSP-k	Top-k selection problem
$\vec{a} = (a_1, a_2, \ldots, a_d)$	Vector or point in R^d
$\vec{f} = (f_1, f_2, \ldots, f_d)$	Preference vector associated with linear f; $f(\vec{a}) = \vec{f} \circ \vec{a}$
$TSP_k(\vec{f})$	List of top k tuples in S, in decreasing order of scores wrt $\vec{f}$
TSI-k	Top-k Selection Index; represents a polyhedral division of R_+^d
$L(R)$	List of top-k tuples associated with region $R \subseteq R_+^d$

Top-k Selection Problem (TSP-k): Given a user preference function f, find the k tuples in S with the largest scores according to f. Let $TSP_k(f)$ denote the output (an ordered list) consisting of these k tuples, in nonincreasing order of scores.

Definition 3.2. [domination $\preccurlyeq$, incomparable, sdomination$\prec$]

Let $t,\ t' \in S$.

- t *is dominated by* t', denoted by $t \preccurlyeq t'$, if $A_i(t) \preccurlyeq A_i(t')$ for all i.
- t and t' are *incomparable*, if $t \not\preccurlyeq t'$ and $t' \not\preccurlyeq t$.
- Let $t \preccurlyeq t'$. t *is strictly dominated* (*or sdominated*) *by* t', denoted by $t \prec t'$, if $A_i(t) < A_i(t')$ for at least one i. ◊

Definition 3.3. [monotone]

A preference function f is *monotone* if $f(t) \leqq f(t')$, whenever $t \preccurlyeq t'$. ◊

Let $\vec{a}$ denotes the point $(a_1, a_2, \ldots, a_d) \in R^d$. It can be thought as a vector from the origin to the point.

Definition 3.4. [linear function, associated vector, ∘]

A preference function $f: R_+^d \rightarrow R_+$ is *linear* if $f(\vec{a}) = f_1 a_1 + f_2 a_2 + \cdots + f_d a_d$, for some $f_i \in R (1 \leq i \leq d)$. $\vec{f} = (f_1, f_2, \ldots, f_d) \in R^d$ is the *vector associated with* f, then $f(\vec{a}) = \vec{f} \circ \vec{a}$ where ∘ denotes the inner (dot) product. ◊

Fact 3.1. [linear monotone]

A linear function $f: R_+^d \rightarrow R_+$ is *monotone*, iff $f_i \geq 0$ for all i; i.e., $\vec{f} \in R_+^d$. ◊

We consider the TSP-k problem described above, with only *linear monotone preference functions*.

Top-k Selection Index (TSI-k): Given a relation S, and an integer $k \geq 1$, construct an index that can be used to solve TSP-k. ◊

Our approach for index construction is as follows. Consider the space R_+^d of all possible preference vectors $\vec{f}$. We divide R_+^d into some number $NR(n,k)$ of convex polyhedral regions; TSI-k represents this division. With each such region R, we associate an ordered list $L(R)$ of top-k tuples from S; for all $\vec{f} \in R$, $TSP_k(\vec{f}) = L(R)$. The lists associated with different regions are different.

TABLE 3

DESIRED BOUNDS ON PERFORMANCE MEASURES

Measure	**Description**	**Desired Bound**
$NR(n,k)$	Number of polyhedral regions in TSI-k	$O(kn)$
$PT(n,k)$	Time used to construct TSI-k	$O(dn\log n + k^2 n)$
$IS(n,k)$	space required to store TSI-k	$O(k^2 n)$
$QT(n,k)$	Time to answer a query	$O(d\log n + k)$

Each region R in TSI-k is a convex polyhedral region for the following reason. R consists of all $\vec{f}$ that satisfy a set of linear inequalities. There are two kinds of such inequalities:

- For $t \in L(R)$ and $t' \notin L(R)$, we have $f(t) \geq f(t')$.
- For $t, t' \in L(R)$, t precedes t' in $L(R)$, we have $f(t) \geq f(t')$.

Each inequality defines a *half-space* in R^d, and a half space is convex Preparata and Shamos [62]. R is the intersection of all these half-spaces, and the positive orthant R_+^d; so, R is *convex*. Note that, for a given S, some lists L of k tuples from S would not have an associated region; this is because the interior of the region is empty.

Once we have constructed TSI-k, query processing works as follows. Given a preference vector $\vec{f}$, we only have to locate it in this division, to see which polyhedral region R it lies in; then we just output the associated list $L(R)$. This method uses techniques from computational geometry, both to construct the index and to locate $\vec{f}$. We have the following:

Theorem 3.1.

The TSP-k problem and TSI-k index are scale invariant.

1) $TSP_k(\vec{f})$ depends only on the direction of $\vec{f}$, not its magnitude, i.e., $TSP_k(\alpha \vec{f}) = TSP_k(\vec{f})$, for all constants $\alpha > 0$.
2) For a region R in the index: if $\vec{f} \in R$, then $\alpha \vec{f} \in R$, for all constants $\alpha > 0$.

Proof. Consider the convex polyhedral regions described above. The inequality $f(t) \geq f(t')$ is equivalent to the in-equality $\alpha f(t) \geq \alpha f(t')$, for all constants $\alpha > 0$. ◊

We let $NR(n,k)$ deonte the number of polyhedral regions referred to above, in TSI-k; $PT(n,k)$ denotes the pre-processing time used to construct TSI-k; $IS(n,k)$ denotes its size (space required to store it). $QT(n,k)$ denotes the time to answer a query: given f, output $TSP_k(\vec{f})$.

We consider the case where the relation S and the TSI-k index fit in memory. Disk based solution is a topic for future research. Table 3 presents our expected/desired values for the four performance measures. Note that $IS(n,k)$ includes storing the output list $L(R)$, for each of the $NR(n,k)$ regions; this also explains the k^2 term in $PT(n,k)$. The $+k$ term in $QT(n,k)$ is for output of the top-k tuples. We show that the objective of table 3 can be achieved for $d = 2, k = 2$.

Our approach is closely related to that of Chang et al. [16] and Tsaparas et al. [73], but goes well beyond them, even for $d = 2$. Note that Tsaparas et al. [73] only considers the two dimensional case.

CHAPTER 4

MAXIMAL AND CONVEX LAYERS

Let X be a set of n points in R_+^d. We will discuss the maximal and convex layers of X, and the relationship between them.

Section 4.1. Maximal Layers

A point $p \in X$ is said to be *maximal* if p is not sdominated by any other point in X. Let $maxima(X)$ denote the set of all maximal points in X.

The *maximal layers* of X are defined as follows (Figure 1a):

$ML_1(X) = maxima(X)$,

$ML_2(X) = maxima(X - ML_1(X))$,

and in general $ML_i(X) = maxima\ (X - \bigcup_{j=1}^{i-1} ML_j(X))$

Let m_i denote the number of vertices in $ML_i(X)$; $\sum_i m_i = n$.

In a set of points, *skyline* points are such set of points which are not dominated by any other points in the set. Similar to this, *maximal layer* defines the set of points which are not dominated by any other points in the set. Figure 1a contains seven different points ranging from $P_1, P_2, \ldots$ to P_7 which are obtained by $maxima(X)$. First

Maximal Layer $ML_1(X)$ for this is given by the $maxima(X)$. Hence, $ML_1(X)$ is referred to as the *skyline* of X.

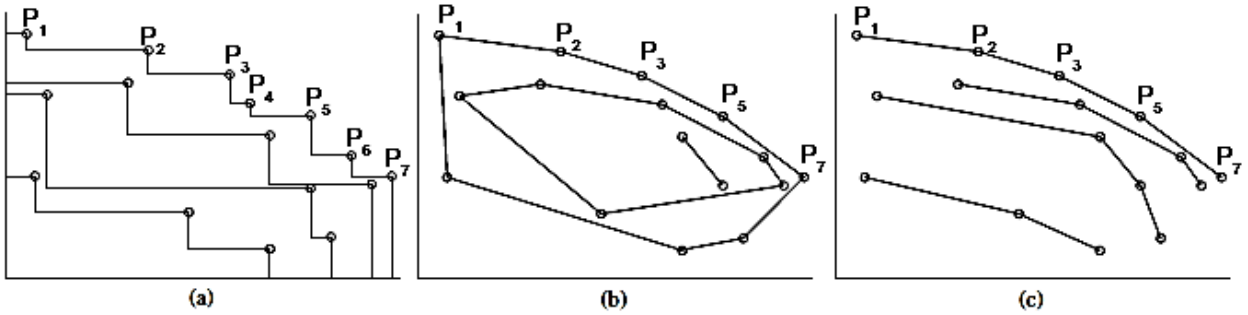

Figure 1: a) Maximal Layers $\boldsymbol{ML_i(X)}$, b)Convex Layers $\boldsymbol{CL_i(X)}$ and c) Upper Convex Layers $\boldsymbol{UCL_i(X)}$

Theorem 4.1.

Let X be a set of n points in R_+^d. $maxima(X)$ can be found out in $O(n \log n + n(\log n)^{d-2})$ time[62]. Further for $d = 2, 3, \ldots$, all the maximal layers can be found in $O(n \log n)$ time [10].◊

Section 4.2. Convex Layers

Let $CH(X)$ denotes the c*onvex hull* of X; $CH(X)$ is the smallest convex polyhedron that contains all the points in X[62].The vertices of $CH(X)$ are some of the points in X.

The *convex layers* of X are defined as follow (Figure 1b):

$CL_1(X) = CH(X)$

$CL_2(X) = CH(X - CL_1(X))$,

and in general,

$CL_i(X) = CH(X - \bigcup_{j=1}^{i-1} CL_j(X))$.

Let $R_{-\infty}^d$ denotes the set of d points at $-\infty$ on the d axes; for example, $R_{-\infty}^2 = \{(-\infty, 0), (0, -\infty)\}$.

Let the *upper convex hull* of X, denoted by $UCH(X)$, be $CH(X \bigcup R_{-\infty}^d) - R_{-\infty}^d$. $UCH(X)$ is obtained from $CH(X \bigcup R_{-\infty}^d)$, by deleting all the points in $R_{-\infty}^d$ and the faces incident on them.

The *upper convex layers* of X is defined as follows (Figure 1c):

$UCL_1(X) = UCH(X)$

$UCL_2(X) = UCH\big(X - UCL_1(X)\big)$,

and in general,

$UCL_i(X) = UCH(X - \bigcup_{j=1}^{i-1} UCL_j(X))$.

Let n_i denotes the number of vertices in $UCL_i(X)$;$\sum_i n_i = n$. N_k denotes $\sum_{i=1}^{k} n_i$.

Theorem 4.2.

Let X be a set of n points in R_+^d.

1) For $d = 2,3$: $UCL_1(X)$ can be found in $O(n \log n)$ time [62].
2) For $d = 2,3$: the first k upper convex layers can be found in $O(kn \log n)$ time [62].
3) For $d = 2$, all the upper convex layers can be found in $O(n \log n)$ time [19].

4) For $d = 3$, all the upper convex layers can be found in $O(n \log^6 n)$ time [14].

5) For $d > 3$: $UCL_1(X)$ can be found in $O(n^{\lfloor \frac{d}{2}+1 \rfloor})$ time [62]. ◊

Our proposed TSI-k index is based on the upper convex layers. The results referred to the theorem 4.2 were originally for convex layers; they can be trivially modified for upper convex layers. In the aforementioned theorem, the algorithms referred to in items (1) and (2) are very simple, old and standard, with efficient implementations available. Note that in item (2), the run-time is $O(n \log n)$, for all constant k.

Theorem 4.3. [relationship between $ML_i(X) and\ UCL_i(X)$]

When viewed strictly as sets of points:

$UCL_1(X) \subseteq ML_1(X)$

$UCL_2(X) \subseteq \big(ML_1(X) \cup ML_2(X)\big) - UCL_1(X)$,

and in general,

$UCL_i(X) \subseteq \bigcup_{j=1}^{i} ML_j(X) - \bigcup_{j=1}^{i-1} UCL_j(X)$.◊

Upper convex layers are the set of points which are subset of maximal layers. In Figure 1a, first maximal layer $ML_1(X)$ contains set of points from X, upper convex layer $UCL_1(X)$ is subset of $ML_1(X)$, Figure 1c. Likewise, second upper convex layer $UCL_2(X)$ will have a set points from first and second maximal layers i.e. $ML_1(X)$ and $ML_2(X)$, excluding that to the first upper convex layer $UCL_1(X)$.

TABLE 4

NOTATIONS USED IN CHAPTER 4

Item	Description
X	Set of n points in R_+^d
$maxima(X)$	Set of all maximal points in X
$ML_i(X)$	ith maximal layer of X
m_i	Number of vertices in $ML_i(X)$; $\sum_i m_i = n$
$m_k = \sum_{i=1}^{k} m_i$	Number of vertices in the first k maximal layers
$CH(X)$	convex hull of X
$CL_i(X)$	ith convex layer of X
$UCH(X)$	Upper convex hull of X
$UCL_i(X)$	ith upper convex layer of X
n_i	Number of vertices in $UCL_i(X)$; $\sum_i n_i = n$
$N_k = \sum_{i=1}^{k} n_i$	Number of vertices in the first k upper convex layers

CHAPTER 5

TSI-k INDEX IN TWO DIMENSIONS

In this chapter, we consider the TSI-k index in two dimensions. We ignore the non-rank attributes, and consider the relation $S(A_1, A_2)$ with n tuples. Tuples $t = (t_1, t_2) \in S$, and preference vectors $\vec{f} = (f_1, f_2)$ can be considered as points in R_+^2.

Section 5.1. Why Convex Layers?

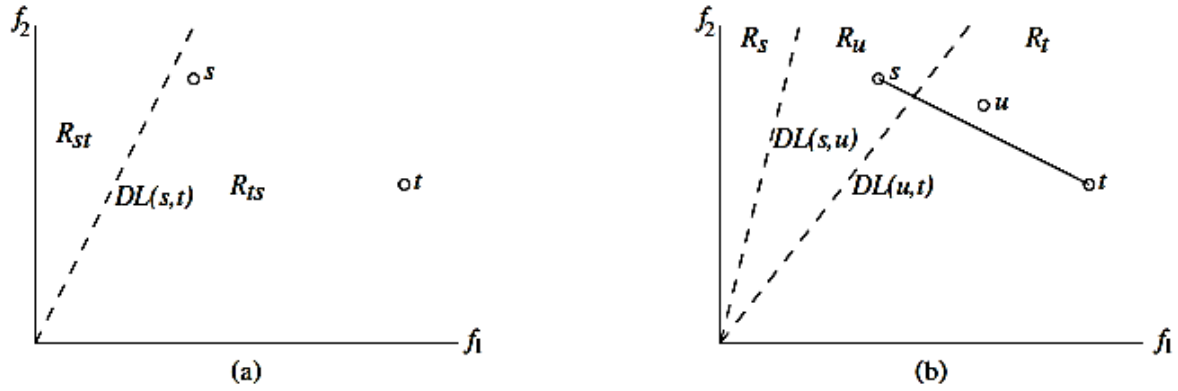

Figure 2: a) Dividing Line $DL(s,t)$ and b) When $R_u \neq \phi$

Consider two tuples $s, t \in S$. If $s \preccurlyeq t$, then $f(s) \leq f(t)$ for all monotone $\vec{f}$. For the case where s and t are incomparable, we have the following.

Definition 5.1. [Dividing Line $DL(s,t)$, regions R_{st} and R_{ts}]

Let $s = (s_1, s_2) \in S$, $t = (t_1, t_2) \in S$, where $s_1 < t_1$ and $s_2 > t_2$ (Figure 2a). The dividing line $DL(s,t)$ is the line defined by the equation $f_1 s_1 + f_2 s_2 = f_1 t_1 + f_2 t_2$; this line

passes through the origin, and is perpendicular to the line segment st [73]. It divides the first quadrant into two regions R_{st} and R_{ts}. For $\vec{f} \in R_{st}$, we have $f_1 s_1 + f_2 s_2 \geq f_1 t_1 + f_2 t_2$; so, s has a better score than t; reverse is true in R_{ts}.◊

Definition 5.2. [angle/slope of a vector/line]

The angle of a vector $\vec{a}$, denoted by $angle(\vec{a})$, is the angle from the positive x-axis to the vector, measured in the counter-clockwise direction. The slope of $\vec{a}$, denoted by $slope(\vec{a})$, is $tan(angle(\vec{a}))$. The angle and slope of a line passing through the origin are defined similarly.◊

Fact 5.1.

Consider two tuples $s = (s_1, s_2) \in S$, $t = (t_1, t_2) \in S$, where $s_1 < t_1$ and $s_2 > t_2$ (Figure 2b). Consider a tuple $u = (u_1, u_2) \in S$ such that $s_1 < u_1 < t_1$. Then u has a better score than both s and t, for some preference function, iff u is above the line segment st.

Proof. If $u_2 \leq t_2$, then $u \prec t$; u is below the line segment st, and $\vec{f} \circ u \leq \vec{f} \circ t$ for all $\vec{f}$; so, let $u_2 > t_2$.

If $u_2 \geq s_2$, then $s \prec u$; u is above the line segment st. Also, $\vec{f} \circ u \geq \vec{f} \circ s$ for all $\vec{f}$; so u has better score than both s and t, to the left of the dividing line $DL(u, t)$; so, let $u_2 < s_2$.

Now, we have $s_1 < u_1 < t_1$, and $s_2 > u_2 > t_2$ (Figure 3b).Consider the region $R_u \subset R_+^2$, consisting of vectors $\vec{f}$ such that $\vec{f} \circ u > \vec{f} \circ s$ and $\vec{f} \circ u > \vec{f} \circ t$.

The first inequality is satisfied in the region R_{us} to the right of the dividing line $DL(s,u)$.

TABLE 5

NOTATIONS USED IN CHAPTER 5

Item	**Description**
$S(A_1, A_2)$	Relation with n tuples; ignore nonrank attributes
$DL(s,t)$	Dividing line for tuples $s,t \in S$
$angle(\vec{a})$	Angle from positive x axis to $\vec{a}$
$slope(\vec{a})$	Slope of $\vec{a}$

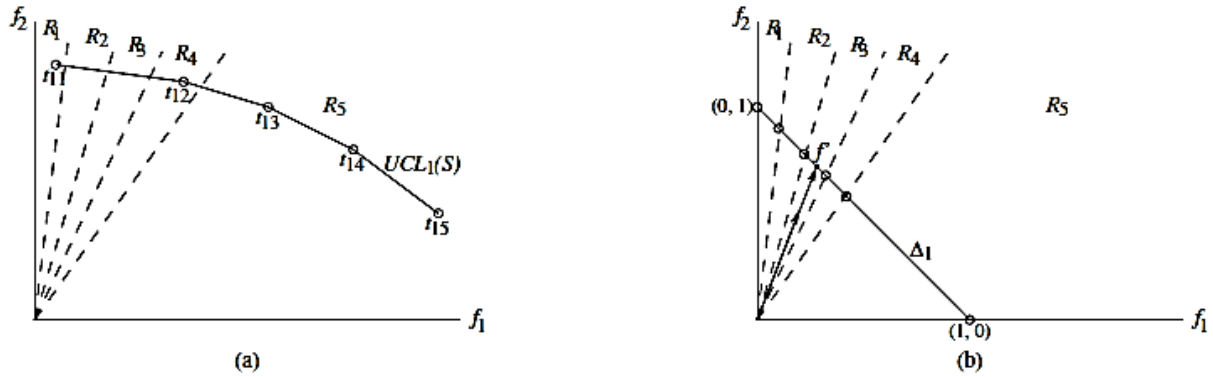

Figure 3: a) The TSI-**1** in 2-dimensions and b) Corresponding pieces of Δ_1

The second inequality is satisfied in the region R_{ut} to the left of the dividing line $DL(u,t)$.

So, $R_u = R_{us} \cap R_{ut}$ is nonempty, iff $DL(s,u)$ is to the left of $DL(u,t)$;

i.e., $slope(DL(s,u)) > slope(DL(u,t))$.

This gives the inequality

$\frac{u_1-s_1}{s_2-u_2} > \frac{t_1-u_1}{u_2-t_2}$ (Eq. 1)

Now, consider the line passing through s and t. Its equation is

$\frac{y-s_2}{x-s_1} = \frac{t_2-s_2}{t_1-s_1}$,

or $y = -\frac{(s_2-t_2)x}{t_1-s_1} + s_2 + \frac{s_1(s_2-t_2)}{t_1-s_1}$ (Eq. 2)

A point (v_1, v_2) lies above this line, iff when we substitute (v_1, v_2) for (x, y) in Eq. 2, we find that $LHS > RHS$. When we substitute (u_1, u_2) for (x, y), the $LHS > RHS$ condition becomes equivalent to Eq. 1.◊

Intuitively, Fact 5.1 means the following. Consider the point p_4 (representing a tuple in S) in Figure 1a. It is below the line segment p_3p_5. So, by Fact 5.1, for *any* f, either p_3 or p_5 would have a better score than p_4. So, p_4 cannot be the top-1 query result for *any* f, even though it is on $ML_1(S)$; note that such points do *not* lie on $CL_1(S)$ or $UCL_1(S)$. This leads to the following.

Theorem 5.1.

A tuple $t \in S$ can be a top-1 query result, for *some* f , iff $t \in UCL_1(S)$.

Our TSI-k index is based on the upper convex layers of S: $UCL_i(S) = (t_{i1}, t_{i2}, \dots, t_{in_i})$. By Theorem 4.2, all these layers can be found in $O(n \log n)$ time. ◊

Section 5.2. The TSI-1 Index

By Theorem 5.1, the TSI-1 index is determined by the tuples in $UCL_1(S) = (t_{11}, t_{12}, \dots, t_{1n_1})$.

The first quadrant R_+^2 is divided into n_1 regions $R_j, 1 \le j \le n_1; L(R_j) = (t_{1j})$ (Figure 3a).

The boundary between regions R_j and R_{j+1} is the dividing line $DL(t_{1j}, t_{1(j+1)})$.

The regions R_j can be constructed from $UCL_1(S)$ in $O(n_1)$ time; the TSI-1 index consists of the slopes of the dividing lines, in increasing order.

Now, consider query processing: Given $\vec{f} \in R_+^d$, output $TSP_1(\vec{f})$.

If $\vec{f} \in R_j$, then $TSP_1(\vec{f}) = \{t_{1j}\}$. Given $\vec{f}$, we can find the region R_j containing it in $O(\log n_1)$ time; this is achieved by binary search for $slope(\vec{f})$, in the TSI-1 index. So, for the TSP-k problem with $k = 1$,

we have,

$NR(n,1) = n_1$,$PT(n,1) = O(n \log n)$, $IS(n,1) = O(n_1)$ and $QT(n,1) = O(\log n_1)$.

Section 5.3. The TSI-2 Index

Now, we want to further subdivide the above regions to answer top-2 queries. Consider a particular region $R_j, 1 < j < n_1$ (Figure 4a).

For a preference vector $\vec{f}$ close to the left boundary $DL(t_{1(j-1)}, t_{1j})$, we have $TSP_2(\vec{f}) = (t_{1j}, t_{1(j-1)})$.

Similarly, for $\vec{f}$ close to the right boundary $DL(t_{1j}, t_{1(j+1)})$, we have $TSP_2(\vec{f}) = (t_{1j}, t_{1(j+1)})$.

Now consider $\vec{f}$ in R_j away from the left and right boundaries; what is the second tuple in $TSP_2(\vec{f})$?

This is where the second layer $UCL_2(S)$ comes in.

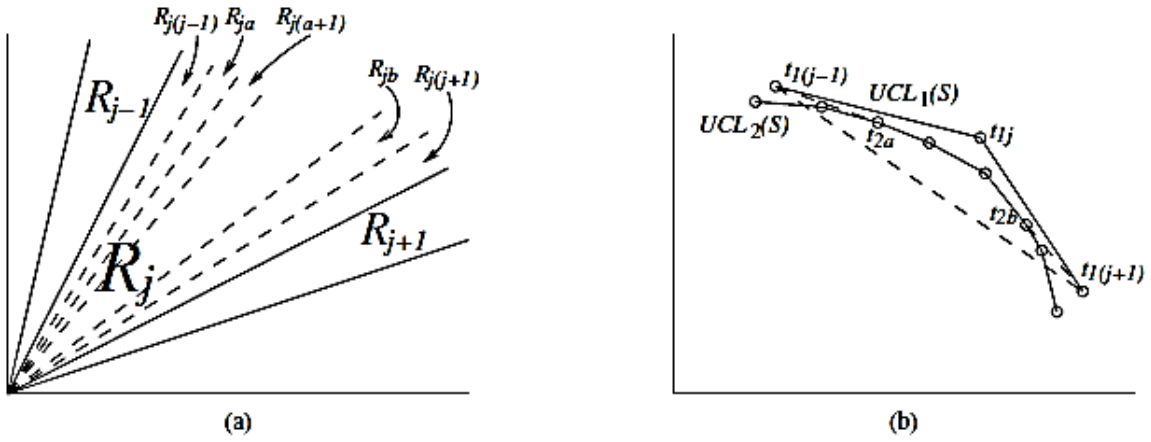

Figure 4: a) Subdivision of $\boldsymbol{R_j}$ and b) part of $\boldsymbol{UCL_2(S)}$ near $\boldsymbol{t_{1j}}$

Consider the points p in $UCL_2(S)$ with x coordinates between that of $t_{1(j-1)}$ and $t_{1(j+1)}$ (Figure 4b).

By Fact 5.1, p can compete with $t_{1(j-1)}$ and $t_{1(j+1)}$ for inclusion in $TSP_2(\vec{f})$ (for some $\vec{f} \in R_j$), only if p is above the segment $t_{1(j-1)}t_{1(j+1)}$; so, consider only the chain C_j consisting of such points.

Consider the tangents $t_{1(j-1)}t_{2a}$ and $t_{1(j+1)}t_{2b}$ to C_j .

Note that $CH(S - \{t_{1j}\})$ can be obtained from $CH(S)$ by replacing t_{1j} with $(t_{2a}, t_{2(a+1)}, \ldots t_{2b})$. So, by Fact 5.1, as we move $\vec{f}$ in R_j , from the left boundary to the right boundary, the second point in $TSP_2(\vec{f})$ moves from left to right of the list $C_j' = (t_{1(j-1)}, t_{2a}, t_{2(a+1)}, \ldots, t_{2b}, t_{1(j+1)})$.

So, we can divide R_j by drawing the dividing lines between each pair of *adjacent* points in C_j' ; all such dividing lines will lie within R_j .

TSI-2 consists of these subdivisions (over all the R_js). For each region R within R_j, $L(R) = (t_{1j}, t)$, where $t \in C_j'$ is the point corresponding to R. Query processing consists of locating the given $\vec{f}$, in a particular region R in TSI-2, using binary search, and outputting the associated $L(R)$.

Now, let us see how to efficiently construct this subdivision of all R_js; the pseudo code for constructing TSI-2 appears in Figure 5.

First, for each point p in $UCL_1(S)$, we need to find the unique point $q \in UCL_2(S)$ to the right of p, such that pq is tangent to $UCL_2(S)$.

For a particular pair (p, q), pq is tangent to $UCL_2(S)$, iff both the points adjacent to q in $UCL_2(S)$ lie below the line pq; this can be tested in $O(1)$ time. As we start at the left end of $UCL_1(S)$, and move p to the right, the point $q \in UCL_2(S)$ it is tangent at also moves to the right. So, we can find all tangents (p, q) by moving to the right in both $UCL_1(S)$ and $UCL_2(S)$, in $O(n_1 + n_2)$ time.

Similarly, for each point p in $UCL_1(S)$, we can find the unique point $q \in UCL_2(S)$ to the left of p, such that pq is tangent to $UCL_2(S)$.

Now, for each R_j, we can construct the list C_j' , and do the subdivision in $O(|C_j'|)$ time. The number of subdivisions in R_j is exactly $|C_j'|$ .C_j' and C_{j+1}' can share at most one point, as follows.

$C_j' = (t_{1(j-1)}, t_{2a}, t_{2(a+1)}, \ldots, t_{2b}, t_{1(j+1)})$

$C_{j+1}' = (t_{1j}, t_{2c}, t_{2(c+1)}, \ldots, t_{2d}, t_{1(j+2)})$

The only *possible* overlap between these two lists is that $t_{2b} = t_{2c}$. So, the total number of subdivisions over all $R_j, 1 \leq j \leq n_1$, is at most $2n_1 + n_2 + \min(n_1, n_2)$; the term $2n_1$ accounts for the first and last term in each C_j'; the second term n_2 accounts for the other terms in the C_j's, excluding possible overlaps, if they together contain all the points in $UCL_2(S)$; the third term $\min(n_1, n_2)$ accounts for possible overlap.

Note that $2n_1 + n_2 + \min(n_1, n_2) = (n_1 + n_2) + (n_1 + \min(n_1, n_2)) \leq n + n = 2n$; so, $NR(n, 2) \leq 2n$.

TSI-2 consists of these subdivisions; with each subdivision R, we store $L(R) = TSP_2(\vec{f})$ for all $\vec{f}$ in that subdivision. Query processing consists of locating the given $\vec{f}$ in this subdivision using binary search, and outputting the associated $L(R)$.

Section 5.4. The TSI-k Index

This can be extended to larger k. In particular, we prove below that $NR(n, n) \leq n(n-1)/2$, much less than $n!$. For that, we need the following.

TSI-2 construction in 2-dimensions

Construct the upper convex layers of S (Figure 1c)

Let $UCL_1(S) = (t_{11}, t_{12}, \ldots, t_{1n_1})$

Construct the dividing lines $DL(t_{1j}, t_{1(j+1)}, 1 \leq j \leq n_1)$; they divide the first quadrant into n_1 regions $R_j; 1 \leq j \leq n_1$ (Figure 3a)

The TSI-1 index consists of the slopes of the dividing lines, in increasing order

For each $p \in UCL_1(S)$, find $q \in UCL_2(S)$ to the right of p, such that pq is tangent to $UCL_2(S)$

For each $p \in UCL_1(S)$, find $q \in UCL_2(S)$ to the left of$p$, such that pq is tangent to $UCL_2(S)$

For each $j, 1 \leq j \leq n_1$ do // Want to subdivide R_j (Figure 4a)

// First construct the list C_j' as follows

if $j = 1$ then let $t_{10} \leftarrow (0, y(t_{12}))$

if $j = n_1$ then let $t_{1(n_1+1)} \leftarrow (x(t_{1(n_1-1)}), 0)$

$C_j \leftarrow$ list of (adjacent) points in $UCL_2(S)$ that are above or to the right of segment $t_{1(j-1)}t_{1(j+1)}$

Figure 5: Psuedo-code for construction of TSI-2

else let $t_{2a} \in UCL_2(S)$ be the point to the right of $t_{1(j-1)}$, such that $t_{1(j-1)}t_{2a}$ is tangent to $UCL_2(S)$

let $t_{2b} \in UCL_2(S)$ be the point to the left of $t_{1(j+1)}$, such that $t_{1(j+1)}t_{2b}$ is tangent to $UCL_2(S)$

let $C_j' = (t_{1(j-1)}, t_{2a}, t_{2(a+1)}, \dots, t_{2b}, t_{1(j+1)})$ (Figure 4b)

if $j = 1$ then drop the first point t_{10} from C_j'

if $j = n_1$ then drop the last point $t_{1(n_1+1)}$ from C_j'

Divide R_j by drawing the dividing lines between each pair of adjacent points in C_j'

Store the dividing lines in increasing order of their slopes

For each subdivision within R_j , associate the result list (t_{1j}, t),

where $t \in C_j'$ is the point corresponding to the subdivision.

Figure 5 (contd.)

TABLE 6

COMPARISON OF OUR RESULTS WITH [73] AND [16]

Measure	**Ours**	**[73]**	**[16]**
$NR(n,k)$	$O(k\|D_K\|)$	$O(k\|D_K\|)$	1
$PT(n,k)$	$O(n\log n + k^2\|D_K\|)$	$O(\|D_K\|^2\log\|D_K\|)$	$O(n\ \log n)$
$IS(n,k)$	$O(k^2\|D_K\|)$	$O(k^2\|D_K\|)$	$O(\|D_k\|)$
$QT(n,k)$	$O(log\|D_K\| + k)$	$O(log\|D_K\| + k\log k)$	$O(\|D_k\|)$

Definition 5.3. [$inversions(\pi)$]

Let π be a permutation of $(1,2,\dots,n)$. An inversion in π is a pair of integers $(i,j), 1 \le i < j \le n$, such that j appears before i in π. Let $inv(\pi)$ denote the set of all inversions in π; it uniquely determines π. ◊

For example, $inv(2413) = \{(1,2),(1,4),(3,4)\}$. For identity permutation $\pi_I = 12\dots n, inv(\pi_I) = \phi$.

For $\pi = n(n-1)\dots 1, inv(\pi)$ contains all pairs $(i,j), 1 \le i < j \le n$; so, $|inv(\pi)| = n(n-1)/2$, the maximum possible.

Theorem 5.2.

$NR(n,n) \leq n(n-1)/2$.

Proof. Consider a set $S = \{t_1, t_2, \ldots, t_n\}$ of tuples. Without loss of generality, suppose that these tuples are in nonincreasing order of y coordinate values; i.e., $y(t_1) \geq y(t_2) \geq \ldots \geq y(t_n)$.

Let π_x and π_y denote the ordering of (the indices of) these tuples in non-increasing order of their x and y coordinate values, respectively. So, π_x and π_y are permutations of $(1,2,\ldots,n)$. By our assumption above,

$\pi_y = 12 \ldots n; inv(\pi_y) = \phi$.

Consider the subdivision of R_+^2 corresponding to TSI-n. For a region R, let $\pi(R)$ be the permutation consisting of the indices of the tuples in $L(R)$, in that order. For the leftmost region R_1 adjacent to the y axis, $\pi(R_1) = \pi_y = 12 \ldots n$. For the rightmost region R_N adjacent to the x axis, $\pi(R_N) = \pi_x$.

Consider a region R_i and the next region R_{i+1} to its right. The dividing line between these regions is $DL(t_a, t_b), 1 \leq a < b \leq n$. In $L(R_i)$, t_a appears just before t_b; i.e., a appears just before b in $\pi(R_i)$. In $L(R_{i+1})$, t_b appears just before t_a; i.e., b appears just before a in $\pi(R_{i+1})$. So, as we move from R_i toR_{i+1}, we add exactly one inversion (a, b) to the permutation; as we continue to the right, this inversion is never reversed, because t_b has a higher score than t_a in the half-space to the right of

$DL(t_a, t_b)$. So, the total number of regions N is the number of inversions we add, as we move from R_1 to R_N; this is exactly

$inv\big(\pi(R_N)\big) = inv(\pi_x) \leq n(n-1)/2.\Diamond$

This can be extended to general k, to show that $N(n,k) = O(kn)$. We have the following.

Theorem 5.3.

The performance bounds listed in Table 2 can be achieved for two dimensions.$\Diamond$

Comparison of Our Result with Tsaparas et al. [73].

Among all the previous works, their Ranked Join Index is closest to our TSI-k index. They first select a set $D_k \subseteq S$ of tuples; no tuple in D_k is dominated by at least k tuples in S. $D_k \subseteq \bigcup_{i=1}^{k} ML_i(S)$; so $|D_k| \leq M_k$. Table 5 presents the various performance measures, for our algorithm and theirs, in terms of $|D_k|$. Our preprocessing time is much less.

Comparison of Our Result with Chang et al. [16].

For query processing, Chang et al. [16] computes and stores the upper convex layers. They do not have an index similar to our TSI-k index. Their query processing is expensive. Table 5 presents the various performance measures.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We use geometrical computation, namely *convex layers*, to output top-k tuples using user preference functions for a relation. Our approach consists of two steps; index construction and query processing. At the time of index construction, relation is known but preference function is not known. We restrict this function to the class of linear monotone functions. Given this preference function, we use the index to output the top-k tuples using computational geometry. For carrying out both of these processes efficiently, convex and maximal layer is used to determine the dominating sets which results in the output of query from top-k selection index. We have restricted our findings for top-k ($k = 2$) in two dimensions ($d = 2$). This can be further extended to general k and general d ($k = n$, $d = n$).

REFERENCES

REFERENCES

[1] D. J. Abraham, K. Cechlarova, D. Manlove and K. Mehlhorn. Pareto Optimality in House Allocation Problems, *ISAAC*, 2005, pp. 1163–1175.

[2] W. Balke, U. Glzntzer and J. Zheng. Efficient Distributed Skylining for Web Information Systems, *Proc. Intl. Conf. Extending Database Tech.* (*EDBT*), 2004, pp. 256–273.

[3] W. T. Balke, W. Siberski and U. Glzntzer. Getting Prime Cuts from Skylines over Partially Ordered Domains, *BTW*, 2007, pp. 64–81.

[4] I. Bartolini, P. Ciaccia and M. Patella. Efficient Sort-based Skyline Evaluation, *ACM Trans. Database Systems* (*TODS*) **33** (2008), pp. 1–49.

[5] J. L. Bentley, K. L. Clarkson and D. B. Levine. Fast Linear Expected-time Algorithms for Computing Maxima and Convex Hulls. In *SODA*, 1990.

[6] J. L. Bentley, H. T. Kung, M. Schkolnick and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications, *J. ACM* **25** (1978), pp. 536-543.

[7] D. Bertsimas and J. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific, 1997.

[8] S. Borzsonyi, D. Kossmann and K. Stocker. The Skyline Operator, *Proc. IEEE Intl. Conf. Data Engg.* (*ICDE*), 2001, pp. 421–430.

[9] N. Bruno, S. Chaudhuri and L. Gravano. Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation. *ACM Trans. Database Systems* (*TODS*) **27** (2002).

[10] A. L. Buchsbaum and M. T. Goodrich. Three Dimensional Layers of Maxima, Springer-Verlag LNCS 2461/2002, pp. 409–416.

[11] C. Y. Chan, P. K. Eng and K. L. Tan. Stratified Computation of Skylines with Partially-ordered Domains, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2005.

[12] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. Tung and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space, Proc. *ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2006.

[13] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. Tung and Z. Zhang. On High Dimensional Skylines, *Proc. Intl. Conf. Extending Database Tech.* (*EDBT*), 2006, pp. 478–495.

[14] T. M. Chan. A Dynamic Data Structure for 3-D Convex Hulls and 2-D Nearest Neighbor Queries, *J. Assoc. Comput. Mach.* **57** (2010), pp. 16:1–16:15.

[15] K. C. Chang and S. Hwang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2001, pp. 346–357.

[16] Y. C. Chang, L. D. Bergman, V. Castelli, C. S. Li, M. L. Lo and J. R. Smith. The Onion Technique: Indexing for Linear Optimization Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2000, pp. 391–402.

[17] S. Chaudhuri, N. Dalvi and R. Kaushik. Robust cardinality and cost estimation for skyline operator, *Proc. IEEE Intl. Conf. Data Engg.* (*ICDE*), 2006.

[18] S. Chaudhuri and L. Gravano. Evaluating Top-k Selection Queries, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 1999, pp. 397–410.

[19] B. Chazelle. On Convex Layers of a Planar Set, *IEEE Trans. Inform. Theory* **31** (1985), pp. 509–517.

[20] L. Chen and X. Lian. Dynamic Skyline Queries in Metric Spaces, *Proc. Intl. Conf. Extending Database Tech.* (*EDBT*), 2008.

[21] J. Chomicki. Preference Formulas in Relational Queries, *ACM Trans. Database Systems* (*TODS*) **28** (2003), pp. 427–466.

[22] J. Chomicki, P. Godfrey, J. Gryz and D. Liang. Skyline with Presorting, *Proc. IEEE Intl. Conf. Data Engg.* (*ICDE*), 2003, pp. 717-728. Also, *Intelligent Information Systems* (2005), pp. 595–604.

[23] J. L. Cohon. *Multiobjective Programming and Planning*. Dover Publications, 2004.

[24] D. L. Craft, T. F. Halabi, H. A. Shih and T. R. Bortfeld. Approximating Convex Pareto Surfaces in Multiobjective Radiotherapy Planning, *Med. Phys.* **33** (2006), pp. 3399–3407.

[25] G. Das, D. Gunopulos, N. Koudas and D. Tsirogiannis. Answering Top-k Queries using Views, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2006, pp. 451–462.

[26] E. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2007, pp. 291–302.

[27] D. Douglas and T. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature, *The Canadian Cartographer* **10** (1973), pp. 112–122.

[28] H. Garcia-Molina, J. D. Ullman and J. Widom. *Database Systems*. Prentice Hall, NJ, 2009.

[29] P. Godfrey. Skyline Cardinality for Relational Processing, Found. *Inform. and Knowledge Syst. Conf* (*FoIKS*), 2004, pp. 78–97.

[30] P. Godfrey, R. Shipley and J. Gryz. Algorithms and Analyses for Maximal Vector Computation, *VLDB J* **16** (2007), pp. 5–28. Also, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2005, pp. 229–240.

[31] M. Goncalves and M. E. Vidal. Top-k Skyline: A Unified Approach*, OTM Workshops*, 2005.

[32] P. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms, 1997.

[33] V. Hristidis, N. Koudas and Y. Papakonstantinou. Prefer: A System for the Efficient Execution of Multi-parametric Ranked Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2001, pp. 259–270.

[34] V. Hristidis and Y. Papakonstantinou. Algorithms and Applications for Answering Ranked Queries using Ranked Views, *VLDB J* **13** (2004), pp. 49–70.

[35] S. Hwang and K. C. Chang. Optimizing Top-k Queries for Middleware Access: A Unified Cost-based Approach, *ACM Trans. Database Systems* (*TODS*) **32** (2007).

[36] I. F. Ilyas, W. Aref and A. K. Elmagarmid. Supporting Top-k Join Queries in Relational Databases, *VLDB J* **13** (2004), pp. 207–221.

[37] I. F. Ilyas, G. Beskales and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems, *ACM Comput. Surv.* **40** (2008), pp. 11:1–11:58.

[38] B. Jiang, J. Pei, X. Lin, D. Cheung and J. Han. Mining Preferences from Superior and Inferior Examples, *Proc. Conf. Knowledge Discovery and Data Mining* (*KDD*), 2008, pp. 390–398.

[39] M. E. Khalefa, M. F. Mokbel and J. J. Levandoski. Skyline Query Processing for Incomplete Data, *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2008.

[40] W. Kiessling. Foundations of Preferences in Database Systems, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2002, pp. 311–322.

[41] M. Kontaki, A. N. Papadopoulos and Y. Manolopoulos. Continuous Top-k Dominating Queries in Subspaces, *Panhellenic Conference on Informatics*, 2008.

[42] F. Korn, B. U. Pagel and C. Faloutsos. On the Dimensionality Curse and the Self Similarity Blessing, *IEEE Trans. Know. Data Engg*. (*TKDE*) **13** (2001), pp. 96–111.

[43] D. Kossmann, F. Ramsak and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2002, pp. 275–286.

[44] J. Lee, G.W. You and S.W. Hwang. Personalized Top-k Skyline Queries in High-dimensional Space, *Inform. Syst.* **34** (2009), pp. 45–61.

[45] K. Lee, B. Zheng, H. Li andW. Lee. Approaching the Skyline in z Order, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2007, pp. 279–290.

[46] C. Li, B. Ooi, A. K. Tung and S. Wang. Dada: A Data Cube for Dominant Relationship Analysis, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2006, pp. 659–670.

[47] C. Li, A. Tung, W. Jin and M. Ester. On Dominating Your Neighborhood Profitably, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2007, pp. 818–829.

[48] X. Lian and L. Chen. Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2008.

[49] X. Lian and L. Chen. Top-k Dominating Queries in Uncertain Databases, *Proc. Intl. Conf. Extending Database Tech.* (*EDBT*), 2009, pp. 660–671.

[50] X. Lin, Y. Yuan, Q. Zhang and Y. Zhang. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows, *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2005, pp. 502–513.

[51] X. Lin, Y. Yuan, Q. Zhang and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator, *Proc. IEEE Intl. Conf. Data Engg.* (*ICDE*), 2007, pp. 86–95.

[52] D. Mindolin and J. Chomicki. P-Skyline Preference Relations, http://mindolin.info/pskyline/. Accessed September 2010.

[53] D. Mindolin and J. Chomicki. Discovering Relative Importance of Skyline Attributes, *Proc. VLDB endowment* **2** (2009), pp. 610–621.

[54] M. D. Morse, J. M. Patel and W. I. Grosky. Efficient Continuous Skyline Computation, *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2006.

[55] M. D. Morse, J. M. Patel and H. V. Jagadish. Efficient Skyline Computation over Low-cardinality Domains, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2007, pp. 267–278.

[56] K. Mouratidis, S. Bakiras and D. Papadias. Continuous Monitoring of Top-k Queries over Sliding Windows, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2006, pp. 635–646.

[57] D. Nanongkai, A. Sarma, A. Lall, R. J. Lipton and J. Xu. RegretMinimizing Representative Databases, *Proc. VLDB Endowment* (*PVLDB*) **2** (2009).

[58] D. Papadias, Y. Tao, G. Fu and B. Seeger. Progressive Skyline Computation in Database Systems, *ACM Trans. Database Systems* (*TODS*) **30** (2005), pp. 41–82.

[59] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos and Y. Manolopoulos. Domination Mining and Querying, *Proc. Intl. Conf. Data Warehousing and Knowledge Discovery* (*DaWaK*), 2007.

[60] J. Pei, B. Jiang, X. Lin and Y. Yuan. Probabilistic Skylines on Uncertain Data, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2007, pp. 15–26.

[61] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu and Q. Zhang. Towards Multidimensional Subspace Skyline Analysis, *ACM Trans. Database Systems* (*TODS*) **31** (2006), pp. 1335–1381.

[62] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[63] U. Ramer. An Iterative Procedure for the Polygonal Approximation of Plane Curves, *Computer Graphics and Image Processing* **1** (1972), pp. 244–256.

[64] G. Rote. The Convergence Rate of the Sandwich Algorithm for Approximating Convex Functions, *Computing* **48** (1992), pp. 337–361.

[65] D. Sacharidis, S. Papadopoulos and D. Papadias. Topologically sorted skylines for partially ordered domains. *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2009.

[66] N. Sarkas, G. Das, N. Koudas and A. Tung. Categorical Skylines for Streaming Data, Proc. *ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2008, pp. 239–250.

[67] A. Sarma, A. Lall, D. Nanongkai and J. Xu. Randomized Multipass Streaming Skyline Algorithms, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2009.

[68] K. L. Tan, P. K. Eng and B. C. Ooi. Efficient Progressive Skyline Computation, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2001, pp. 301–310.

[69] Y. Tao, L. Ding, X. Lin and J. Pei. Distance-based Representative Skyline, *Proc. IEEE Intl. Conf. Data Engg.* (*ICDE*), 2009, pp. 892–903.

[70] Y. Tao, V. Hristidis, D. Papadias and Y. Papakonstantinou. Branch and Bound Processing of Ranked Queries, *Inform. Syst*. **32** (2007), pp. 424–445.

[71] Y. Tao and D. Papadias. Maintaining Sliding Window Skylines on Data Streams, *IEEE Trans. Know. Data Engg.* (*TKDE*) **18** (2006), pp. 377–391.

[72] Y. Tao, X. Xiao and J. Pei. Subsky: Efficient computation of skylines in subspaces. *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2006.

[73] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas and D. Srivastava. Ranked Join Indices, *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2003.

[74] L. H. U, N. Mamoulis and K. Mouratidis. Efficient Evaluation of Multiple Preference Queries, *Proc. IEEE Intl. Conf. Data Engg*. (*ICDE*), 2009, pp. 1251–1254.

[75] L. H. U, N. Mamoulis and K. Mouratidis. A Fair Assignment Algorithm for Multiple Preference Queries, *Proc. Intl. Conf. Very Large Data Bases* (*VLDB*), 2009.

[76] L. H. U, N. Mamoulis and M. L. Yiu. Computation and Monitoring of Exclusive Closest Pairs, *IEEE Trans. Know. Data Engg*. (*TKDE*), to appear.

[77] L. H. U, M. L. Yiu, K. Mouratidis and N. Mamoulis. Capacity Constrained Assignment in Spatial Databases, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2008, pp. 15–28.

[78] A. Vlachou, C. Doulkeridis and Y. Kotidis. Angle-based Space Partitioning for Efficient Parallel Skyline Computation, *Proc. ACM SIGMOD Intl. Conf. Management of Data* (*SIGMOD*), 2008