Ranking with Uncertain Scoring Functions: Semantics and Sensitivity Measures

Mohamed A. Soliman[*]
Greenplum
San Mateo, USA
mohamed.soliman@emc.com

Ihab F. Ilyas
University of Waterloo
Waterloo, Canada
ilyas@uwaterloo.ca

Davide Martinenghi
Politecnico di Milano
Milano, Italy
davide.martinenghi@polimi.it

Marco Tagliasacchi
Politecnico di Milano
Milano, Italy
marco.tagliasacchi@polimi.it

ABSTRACT

Ranking queries report the top-K results according to a user-defined scoring function. A widely used scoring function is the weighted summation of multiple scores. Often times, users cannot precisely specify the weights in such functions in order to produce the preferred order of results. Adopting uncertain/incomplete scoring functions (e.g., using weight ranges and partially-specified weight preferences) can better capture user's preferences in this scenario.

In this paper, we study two aspects in uncertain scoring functions. The first aspect is the semantics of ranking queries, and the second aspect is the sensitivity of computed results to refinements made by the user. We formalize and solve multiple problems under both aspects, and present novel techniques that compute query results efficiently to comply with the interactive nature of these problems.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Uncertainty, Scoring, Top-k, Ranking, Aggregation

1. INTRODUCTION

Scoring (ranking) functions are among the most common forms of preference specification. A prominent application

---

[*]Work has been done while the author was with University of Waterloo.

```
SELECT R.RestName, R.Street, H.HotelName
FROM RestaurantsInParis R, HotelsInParis H
WHERE distance(R.coordinates, H.coordinates) ≤ 500m
RANK BY w_R· R.Rating + w_H· H.Stars
LIMIT 5
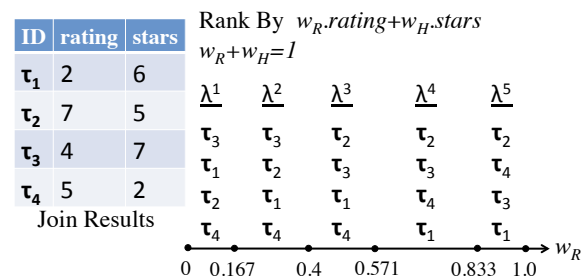```

**Figure 1: A rank join query**



**Figure 2: Possible orderings of the join results**

scenario is joining multiple data sources and ranking join results according to some score aggregation function. The class of queries captured by this scenario is usually referred to as rank join [8], where the objective is to compute the top-$K$ join results based on a given scoring function.

The order of rank join results depends on the chosen score aggregation function. In the simplest but very common case, a linear aggregation function is adopted, which is specified as a weighted sum of scores. For example, Figure 1 shows a rank join query, where Restaurant-Hotel join results are ranked based on a weighted sum of the rating and the number of stars, while reporting only the top 5 join results.

### 1.1 Motivation and Challenges

Often times users cannot precisely specify the weights of the scoring function (e.g., $w_R$ and $w_H$ in Figure 1) in order to produce the preferred order of results. This problem is usually handled either by the user in an interactive trial-and-error manner, or by the machine through learning from user's feedback (e.g., learning weights from user's preference judgment on object pairs [16]). Both approaches have serious limitations. Trial-and-error is a tedious and a time-consuming process that can be very challenging especially to novice users. On the other hand, weight learning requires a

sufficiently large number of user-provided training examples, which can be too demanding.

In many scenarios, adopting an uncertain/incomplete preference specification language can better capture user's preferences. For example, it is much easier for users to provide approximate weights (e.g., ranges) and other hints (e.g., relative importance of the weights) than specifying exact weights. This does not change the goal, which is obtaining a representative ordering of results satisfying user's preferences, but rather provides a flexible and more natural means of preference specification.

Even when a user provides what are believed to be the right weights, it is crucial to analyze the sensitivity of the computed ordering with respect to changes in the given weights. This can provide significant help in data analysis in interactive and exploratory scenarios. For example, when fine-tuning the scoring function, a user may be interested in quantifying the largest change in the weights that does not introduce perturbations in the computed ordering.

We discuss the previous observations using Figure 2, which shows 4 join results for the query in Figure 1.

To illustrate the consequences of uncertain weights, assume that the non-negative weights $w_R$ and $w_H$ are uncertain and normalized to add up to 1. We adopt the convention that results are sorted in descending order of scores. There are 5 possible orderings $\lambda^1, \ldots, \lambda^5$ corresponding to all the possible weight vectors (we only show $w_R$ as $w_H = 1 - w_R$). While $w_R$ is continuous in [0,1], only a finite number of possible orderings exists. These orderings depend on the ranges of $w_R$ and $w_H$ as well as the scores of the join results. Each ordering has properties related to how robust the ordering is wrt. the weight space, and its relationship to other orderings (we give more details in Section 2.2).

To illustrate the importance of quantifying the sensitivity of a specific ordering or weight vector, assume that a user specifies $(w_R, w_H)$ as $(0.16, 0.84)$. The corresponding top-2 results are thus $\langle \tau_3, \tau_1 \rangle$. By changing the weights slightly to $(0.17, 0.83)$ (i.e., $w_R$ increased by only 6%), the top-2 results become $\langle \tau_3, \tau_2 \rangle$. Such sensitivity of the computed ordering to small weight changes may be important to quantify in interactive data analysis.

We advocate the need to model and manage uncertainty in scoring functions. We envision uncertain scoring functions as a means to bridge the gap between imprecise specifications, which are more natural in describing user's preferences, and score-based ranking models, which are based on precise formulations of scoring functions. We identify two important problems pertinent to these settings:

- *Finding a Representative Ordering.* When weights are uncertain, a set of possible orderings is induced. The problem is finding a representative ordering $\lambda^*$ with plausible properties that can be given to the user as an answer. Moreover, when relative preferences among weights are specified (e.g., in Figure 2, the influence of *rating* can be further emphasized by adding the constraint $w_R > w_H$), finding a representative ordering satisfying such preferences is imperative.
- *Quantifying Sensitivity.* When weights are given as input, a corresponding ordering $\lambda$ can be computed. Two sensitivity analysis problems arise. The first problem is quantifying the largest weights change, in the neighborhood of the input weight vector, that does not introduce perturbations in $\lambda$. The second problem

is quantifying the likelihood of obtaining an ordering identical to $\lambda$, given a random weight vector.

## 1.2 Contributions and Paper Organization

Our key contributions are summarized as follows:
- We formulate four novel problems in the context of uncertain scoring functions. We propose multiple query semantics to allow users to reason about the reported answers (Section 2).
- We propose two approaches to compactly represent the set of possible orderings induced by an uncertain scoring function. We give a bound on the number of possible orderings, and present several efficient techniques to compute representative orderings under different semantics (Section 3).
- We introduce a generalization of our methods to handle partially-specified preferences on the scoring function weights (Section 4).
- We present efficient techniques for quantifying the sensitivity of computed results with respect to changes in the input weights (Section 5).

We also conduct an extensive experimental study (Section 6) on real and synthetic data to evaluate the effectiveness and scalability of our solution.

## 2. UNCERTAINTY MODEL AND PROBLEM DEFINITION

Let $\mathbb{R}_+$ represent the set of non-negative real numbers. Consider a set of $d$ relations $R_1, \ldots, R_d$, where each tuple $t_i \in R_i$ includes a score $s(t_i) \in \mathbb{R}_+$. Let $\tau$ be an element of the form $t_1 \bowtie \cdots \bowtie t_d$ taken from the join $\mathcal{O} = R_1 \bowtie \ldots \bowtie R_d$ of the $d$ relations; $\tau$ is referred to as a *join result* [1]. Let $N$ denote the number of join results that can be formed, i.e., $N = |\mathcal{O}|$. The *aggregate score* $\mathcal{F}(\tau)$ of $\tau$ is defined as

$$\mathcal{F}(\tau) = w_1 s(t_1) + \ldots + w_d s(t_d) = \mathbf{w}^T \mathbf{s}(\tau) \qquad (1)$$

where $\mathbf{w} = [w_1, \ldots, w_d]^T$ is a vector in $\mathbb{R}_+^d$, $w_i$ is the weight assigned to the score obtained from relation $R_i$, and $\mathbf{s}(\tau) = [s(t_1), \ldots, s(t_d)]^T$ is the score vector of join result $\tau$.

Given a weight vector $\mathbf{w}$, it is possible to find a total order $\lambda$ of join results in $\mathcal{O}$ based on the aggregate score in (1) (score ties are typically resolved using a deterministic tie-breaker such as tuple IDs). We denote such operation as $\mathcal{O} \overset{\mathbf{w}}{\rightsquigarrow} \lambda$. Note that the order does not change if the weight vector is multiplied by a positive constant. Therefore, without loss of generality, we assume that $\sum_{i=1}^d w_i = 1$.

## 2.1 Uncertainty Model

To model weights uncertainty, let $\mathbf{w}$ be a random variable with probability density function $p(\mathbf{w})$ defined over the standard $(d-1)$-simplex $\Delta^{d-1}$ given by $\Delta^{d-1} = \{\mathbf{w} \in \mathbb{R}_+^d | \sum_{i=1}^d w_i = 1\}$. Hence,

$$\int_{\Delta^{d-1}} p(\mathbf{w}) d\mathbf{w} = 1 \qquad (2)$$

We assume that $p(\mathbf{w})$ is uniform over $\Delta^{d-1}$. Each point on the simplex represents a possible scoring function.

---

[1]Our model is not limited to one score per input relation. We effectively deal with the output (join) relation which may include any possible number of scores.

Let $\boldsymbol{\lambda}_N$ denote a possible ordering of the $N$ join results in $\mathcal{O}$, where $\boldsymbol{\lambda}_N(\tau)$ indicates the position of $\tau$ in $\boldsymbol{\lambda}_N$. The uncertainty in the weights induces a probability distribution on a set of possible orderings $\Lambda_N$, where each $\boldsymbol{\lambda}_N \in \Lambda_N$ occurs with probability $p(\boldsymbol{\lambda}_N)$, computed as follows:

$$p(\boldsymbol{\lambda}_N) = \int_{\mathbf{w} \in \Delta^{d-1}, \mathcal{O} \overset{\mathbf{w}}{\rightsquigarrow} \boldsymbol{\lambda}_N} p(\mathbf{w}) d\mathbf{w} \qquad (3)$$

When the number of join results $N$ is large, we might be interested only in the orderings of $K \leq N$ join results. We denote with $\Lambda_K$ the set of possible top-$K$ answers. Note that each element of $\Lambda_K$ is a prefix of one or more orderings in $\Lambda_N$. Whenever the ordering length is clear from the context, we drop the subscript and write $\boldsymbol{\lambda}$.

## 2.2 Problem Definition

Among the multiple possible ways to construct an ordering from a set of possible orderings, we propose two problem definitions (Problems 2.1 and 2.2) capturing the semantics of representative orderings.

PROBLEM 2.1. [**MPO**] *Given a depth $K$, find the most probable ordering in $\Lambda_K$, defined as $\boldsymbol{\lambda}_{MPO}^* = arg.\max_{\boldsymbol{\lambda} \in \Lambda_K} p(\boldsymbol{\lambda})$.* □

The ordering $\boldsymbol{\lambda}_{\mathrm{MPO}}^*$ is the distribution mode of $\Lambda_K$ (i.e., the ordering that is most likely to be induced by a random weight vector). In Figure 2, for $K = 2$, we have $\boldsymbol{\lambda}_{\mathrm{MPO}}^* = \langle \tau_2, \tau_3 \rangle$, since it corresponds to the largest range of weights.

The next problem definition is based on measuring distance between orderings. The most common among such measures assume orderings with exactly the same elements, and thus cannot be applied to prefixes of orderings.

PROBLEM 2.2. [**ORA**] *Given a distance function $D$, find the optimal rank aggregation of $\Lambda_N$, defined as $\boldsymbol{\lambda}_{ORA}^* = arg.\min_{\boldsymbol{\lambda}} \sum_{\boldsymbol{\lambda}^r \in \Lambda_N} D(\boldsymbol{\lambda}, \boldsymbol{\lambda}^r) \cdot p(\boldsymbol{\lambda}^r)$.* □

We adopt two widely used definitions of the distance function $D$:

- The *Kendall tau distance*, which counts the number of pairwise disagreements in the relative order of items in the two orderings:

$$D(\boldsymbol{\lambda}^r, \boldsymbol{\lambda}^s) = |\{(\tau_i, \tau_j) \in \mathcal{O} \times \mathcal{O} : \boldsymbol{\lambda}^r(\tau_i) < \boldsymbol{\lambda}^r(\tau_j), \boldsymbol{\lambda}^s(\tau_i) > \boldsymbol{\lambda}^s(\tau_j)\}| \qquad (4)$$

- The *Spearman's footrule distance*, which adds up the distance between the ranks of the same item in the two orderings:

$$D(\boldsymbol{\lambda}^r, \boldsymbol{\lambda}^s) = \sum_{\tau \in \mathcal{O}} |\boldsymbol{\lambda}^r(\tau) - \boldsymbol{\lambda}^s(\tau)| \qquad (5)$$

The ordering $\boldsymbol{\lambda}_{\mathrm{ORA}}^*$ is the ordering with the minimum distance summation to all orderings in $\Lambda_N$. In Figure 2, we have $\boldsymbol{\lambda}_{\mathrm{ORA}}^* = \boldsymbol{\lambda}^3$ for either Kendall tau or Spearman's footrule distance.

We next propose formulations of two sensitivity measures: stability of an ordering wrt. weights (Problem 2.3), and ordering likelihood (Problem 2.4).

| Problem | $d = 2$ | $d = 3$ | $d > 3$ |
|---|---|---|---|
| MPO (average case) | $O(N(logN)^{K+1})$ | $O(N(logN)^{2K+1})$ | $O(N^{\lfloor d/2 \rfloor +1}(logN)^{(d-1)K})$ [§] |
| MPO (worst case) | $O(N^2 logN)$ | $O(N^4)$ | $O(N^{2^{d-1}})$ [§] |
| ORA (Kendall tau) | $O(NlogN)$ | NP-Hard | NP-Hard |
| ORA (Footrule) | $O(N^{2.5})$ | $O(N^4)$ | $O(N^{2^{d-1}})$ [§] |
| STB | $O(N)$ | $O(N)$ | $O(dN)$ |
| LIK | $O(N)$ | $O(N^2)$ | $O(N^{2^{d-2}})$ [§] |

[§] Approximate solution.

**Figure 3: Solutions complexity**

PROBLEM 2.3. [**STB**] *Given a depth $K$ and a weight vector $\bar{\mathbf{w}}$, where $\mathcal{O} \overset{\bar{\mathbf{w}}}{\rightsquigarrow} \bar{\boldsymbol{\lambda}}$, find the stability score of $\bar{\mathbf{w}}$, defined as the radius $\rho_K(\bar{\mathbf{w}})$ of the maximal hypersphere $\sigma_K(\bar{\mathbf{w}})$ centered at $\bar{\mathbf{w}}$, such that for all $\mathbf{w} \in \sigma_K(\bar{\mathbf{w}})$, where $\mathcal{O} \overset{\mathbf{w}}{\rightsquigarrow} \boldsymbol{\lambda}$, we have $\boldsymbol{\lambda}_K = \bar{\boldsymbol{\lambda}}_K$.* □

In Problem STB, we compute the largest volume in the weights space, around an input weight vector $\bar{\mathbf{w}}$, in which changing the weights leaves the computed ordering unaltered at least up to depth $K$. In Figure 2, for $\bar{\mathbf{w}} = (0.2, 0.8)$ and $K = 2$, we have $\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^2$. The weight vector $(0.167, 0.833)$ is the furthest vector from $\bar{\mathbf{w}}$ that induces an ordering identical to $\bar{\boldsymbol{\lambda}}$ up to depth 2. Hence, $\sigma_2(\bar{\mathbf{w}})$ is a circle centered at $\bar{\mathbf{w}}$ with $\rho_2(\bar{\mathbf{w}}) = \|(0.2, 0.8) - (0.167, 0.833)\| = 0.047$.

PROBLEM 2.4. [**LIK**] *Given a depth $K$ and a weight vector $\bar{\mathbf{w}}$, where $\mathcal{O} \overset{\bar{\mathbf{w}}}{\rightsquigarrow} \bar{\boldsymbol{\lambda}}_N$, find the likelihood of $\bar{\boldsymbol{\lambda}}_N$ up to depth $K$, defined as $\gamma_K(\bar{\boldsymbol{\lambda}}_N) = \sum_{\boldsymbol{\lambda} \in \Lambda_N, \boldsymbol{\lambda}_K = \bar{\boldsymbol{\lambda}}_K} p(\boldsymbol{\lambda})$.* □

In Problem LIK, we compute the probability of obtaining an ordering identical to $\bar{\boldsymbol{\lambda}}_N$ up to depth $K$. In Figure 2, for $\bar{\mathbf{w}} = (0.5, 0.5)$, we have $\bar{\boldsymbol{\lambda}}_N = \boldsymbol{\lambda}^3$. For $K = 2$, we have $\gamma_2(\boldsymbol{\lambda}^3) = p(\boldsymbol{\lambda}^3) + p(\boldsymbol{\lambda}^4)$, since $\boldsymbol{\lambda}^3$ and $\boldsymbol{\lambda}^4$ are identical up to depth 2.

Figure 3 gives the complexity bounds of our proposed techniques. Our problem instances are configured by three main parameters ($d$, $N$, and $K$) influencing the complexity. We give worst-case complexity bounds for each algorithm. In addition, for Problem MPO, we also give average-case bounds under the assumption of uniformly distributed score vectors. As we show in the next sections, finding ordering probability requires computing a volume in a $d$-dimensional space. For tractability, such volume can only be approximated when $d > 3$.

## 3. REPRESENTATIVE ORDERINGS

One possible approach to compute representative orderings is to *i)* enumerate possible weight vectors, *ii)* find the distinct orderings induced by these vectors, and *iii)* pick the required representative orderings. In addition to being very expensive, such approach can be also inaccurate since it needs to discretize the weights space.

Problem MPO requires processing orderings' prefixes, while Problem ORA requires processing full orderings. Motivated by this observation, we introduce two approaches:

- *A Holistic Approach.* We propose a succinct representation of full orderings as disjoint partitions of the weights space.
- *An Incremental Approach.* We propose a tree-based representation that is incrementally constructed by extending prefixes of orderings.

In general, both approaches can be used for both problems. However, based on our complexity analysis and experiments, we note the superiority of the holistic approach for computing ORA, and the superiority of the incremental approach for computing MPO. Moreover, for the case of $d = 2$, the holistic approach gives rise to useful properties that allow exact and efficient algorithms for both problems.

## 3.1 A Holistic Approach

Our holistic approach leverages the linearity of $\mathcal{F}$ to find, for each pair of join results $(\tau_i, \tau_j)$, a hyper-plane that divides the weights space into two partitions, such that $\mathcal{F}(\tau_i) > \mathcal{F}(\tau_j)$ in one partition, while $\mathcal{F}(\tau_i) < \mathcal{F}(\tau_j)$ in the other partition. By finding all such hyper-planes, we partition the space into disjoint convex polyhedra corresponding to distinct orderings in $\Lambda_N$. In Section 3.1.1, we describe our approach, while in Sections 3.1.2 and 3.1.3, we discuss efficient computation of the MPO and the ORA problems.

### 3.1.1 Possible Orderings Representation

We illustrate our approach for $d = 2$, and then we show how it can be extended to address higher dimensions.

Let $s_{i,n}$ be the $n$-th entry of the score vector $\mathbf{s}(\tau_i)$, where $1 \leq n \leq d$. For $d = 2$, let $\tau_i$ and $\tau_j$ be two distinct join results with score vectors $\mathbf{s}(\tau_i) = [s_{i,1}, s_{i,2}]^T$ and $\mathbf{s}(\tau_j) = [s_{j,1}, s_{j,2}]^T$, respectively. Let $\alpha_{i,j} = p(\mathcal{F}(\tau_i) > \mathcal{F}(\tau_j))$ (i.e., $\alpha_{i,j}$ is the probability of having $\tau_i$ ranked before $\tau_j$). Let $\delta_{i,j,n} = (s_{i,n} - s_{j,n})$. Then, $\alpha_{i,j}$ can be expressed as follow:

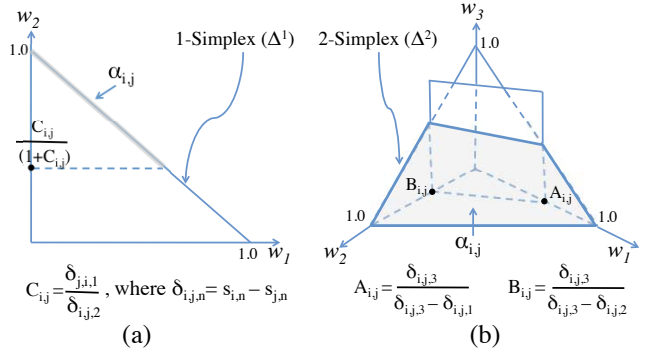$$\alpha_{i,j} = p(w_1 \delta_{i,j,1} + w_2 \delta_{i,j,2} > 0) \quad (6)$$

Due to linearity of $\mathcal{F}(\tau)$, we note that when $\mathbf{s}(\tau_i)$ dominates $\mathbf{s}(\tau_j)$ (i.e., $(s_{i,1} > s_{j,1}$ and $s_{i,2} \geq s_{j,2})$ or $(s_{i,1} \geq s_{j,1}$ and $s_{i,2} > s_{j,2})$), we have $\alpha_{i,j} = 1$. That is, $\mathcal{F}(\tau_i) > \mathcal{F}(\tau_j)$ regardless of the value of the weight vector $\mathbf{w}$. We thus only consider in the following the case where $\mathbf{s}(\tau_i)$ and $\mathbf{s}(\tau_j)$ are incomparable (i.e., $(s_{i,1} > s_{j,1}$ and $s_{i,2} < s_{j,2})$ or $(s_{i,1} < s_{j,1}$ and $s_{i,2} > s_{j,2})$). Lemma 3.1 formulates $\alpha_{i,j}$ in this case:

LEMMA 3.1. *Let* $C_{i,j} = \delta_{j,i,1}/\delta_{i,j,2}$. *Then,* $\alpha_{i,j} = p(w_2 > \frac{C_{i,j}}{1+C_{i,j}})$, *or, equivalently,* $\alpha_{i,j} = p(w_1 < \frac{1}{1+C_{i,j}})$.

PROOF. Since $w_1 + w_2 = 1$, then we have $\alpha_{i,j} = p(w_2(\delta_{i,j,2} - \delta_{i,j,1}) > -\delta_{i,j,1})$. Without loss of generality, we assume that $(\delta_{i,j,2} - \delta_{i,j,1}) > 0$, and hence dividing by $(\delta_{i,j,2} - \delta_{i,j,1})$ does not change the inequality direction (note that if $(\delta_{i,j,2} - \delta_{i,j,1}) < 0$, we can always switch which join result we consider $\tau_i$ and which we consider $\tau_j$). Then, $\alpha_{i,j} = p(w_2 > -\delta_{i,j,1}/(\delta_{i,j,2} - \delta_{i,j,1}))$. Let $C_{i,j} = -\delta_{i,j,1}/\delta_{i,j,2}$, then, by rewriting, we get $\alpha_{i,j} = p(w_2 > \frac{C_{i,j}}{1+C_{i,j}})$. $\square$

Based on Lemma 3.1, a geometrical representation for $\alpha_{i,j}$ for $d = 2$ is given by Figure 4(a), which illustrates that $\alpha_{i,j}$ corresponds to the partition of $\Delta^1$ above the horizontal line $w_2 = C_{i,j}/(1 + C_{i,j})$ [†], while $\alpha_{j,i} = 1 - \alpha_{i,j}$ corresponds to the partition of $\Delta^1$ below $(\tau_i, \tau_j)$. For example, in Figure 5(a), we have $C_{1,4} = (s_{4,1} - s_{1,1})/(s_{1,2} - s_{4,2}) = 3/4$, and hence $\frac{C_{1,4}}{(1+C_{1,4})} = 3/7$ and $\alpha_{1,4} = 4/7$.

---
[†]We only need to consider $\alpha_{i,j} \in (0, 1)$, which implies that $C_{i,j} = (s_{j,1} - s_{i,1})/(s_{i,2} - s_{j,2}) > 0$, and hence $\frac{C_{i,j}}{1+C_{i,j}} \in (0, 1)$.



$C_{i,j} = \frac{\delta_{j,i,1}}{\delta_{i,j,2}}$ , where $\delta_{i,j,n} = s_{i,n} - s_{j,n}$    $A_{i,j} = \frac{\delta_{i,j,3}}{\delta_{i,j,3} - \delta_{i,j,1}}$    $B_{i,j} = \frac{\delta_{i,j,3}}{\delta_{i,j,3} - \delta_{i,j,2}}$

(a)        (b)

**Figure 4: Geometrical representation of** $\alpha_{i,j} = p(\mathcal{F}(\tau_i) > \mathcal{F}(\tau_j))$ **for (a)** $d = 2$ **(b)** $d = 3$

The horizontal line $w_2 = \frac{C_{i,j}}{(1+C_{i,j})}$ is denoted as the *switching line* of $(\tau_i, \tau_j)$, since it separates weights inducing $\mathcal{F}(\tau_i) > \mathcal{F}(\tau_j)$ from weights inducing $\mathcal{F}(\tau_j) > \mathcal{F}(\tau_i)$. When $\alpha_{i,j} = 1$, the switching line of $(\tau_i, \tau_j)$ passes through the point $(0, 0)$ (e.g., the switching line of $(\tau_2, \tau_4)$ in Figure 5(a)).

We show how to extend Lemma 3.1 to handle higher dimensions. By a similar analysis, we conclude that for $d = 3$, the value of $\alpha_{i,j}$ (the shaded area in Figure 4(b)) is the partition of $\Delta^2$ in front of the *switching plane* $w_1(\delta_{i,j,1} - \delta_{i,j,3}) + w_2(\delta_{i,j,2} - \delta_{i,j,3}) + \delta_{i,j,3} = 0$.

In general, the *switching hyper-plane* of $(\tau_i, \tau_j)$ in a $d$-dimensional space is given by $w_1(\delta_{i,j,1} - \delta_{i,j,d}) + w_2(\delta_{i,j,2} - \delta_{i,j,d}) + \cdots + w_{d-1}(\delta_{i,j,d-1} - \delta_{i,j,d}) + \delta_{i,j,d} = 0$. Constructing all switching hyper-planes, for all pairs of join results, divides the weights space into a set of disjoint convex polyhedra, where each minimal convex polyhedron (i.e., a polyhedron that is not enclosed in another polyhedron) corresponds to a distinct ordering in $\Lambda_N$. The reason is that the relative order of any two join results is fixed over all weight vectors inside a minimal polyhedron, while crossing a polyhedron's boundaries changes the relative order of the join results, since these boundaries are parts of the switching planes.

Based on the constraint $\sum_{i=1}^d w_i = 1$, we can reduce the dimensionality of the simplex by eliminating one of the weight variables. This effectively projects the simplex onto a plane with dimensionality smaller by one. For example, for $d = 2$, Figure 5(a) shows the projection of $\Delta^1$ on the $w_2$ axis. There are 5 possible orderings of the shown 4 join results, where each partition of the $w_2$ axis enclosed between two consecutive switching planes (indicated by dotted lines) induces a distinct ordering. Crossing a switching plane changes the relative order of (at least) one pair of join results, and hence a new ordering is induced. Figure 5(b) shows another example for $d = 3$, where we project $\Delta^2$ onto the $w_1 - w_2$ plane, and partition the simplex projection using 2-dimensional switching planes, inducing 7 possible orderings corresponding to 7 minimal convex polygons.

Representing $\Lambda_N$ using switching hyper-planes allows us to derive a bound on the number of possible orderings in $\Lambda_N$, as we formalize in Theorem 3.2.

THEOREM 3.2. *The number of possible orderings in* $\Lambda_N$ *is in* $O(N^{2^{d-1}})$.

PROOF. Given a set of $m$ points in a $d$-dim space, the number of minimal convex polyhedra that can be created
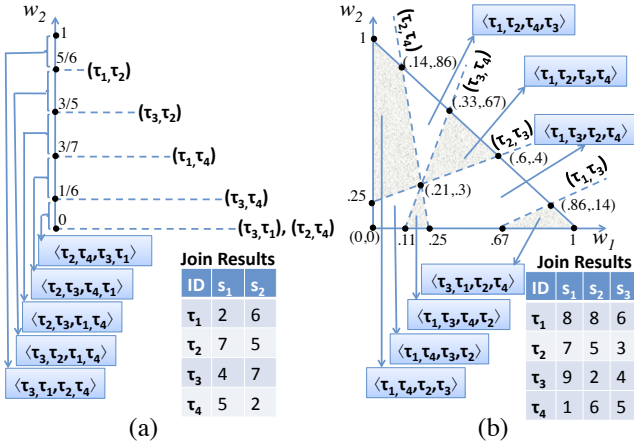
**Figure 5: Switching Planes for (a) $d = 2$ (b) $d = 3$**

is in $O(m)$. Hence, a bound on $|\Lambda_N|$ is the same as the bound on the number of points created by the intersections of $O(N^2)$ hyper-planes, corresponding to pairs of join results, with each other and the simplex. The claim follows from the fact that $O(N^2)$ $(d-1)$-dim planes determine $O(N^{2^{d-1}})$ intersection points, which we show by induction.

Base case ($d = 2$). $\Delta^1$ can be projected onto the 1-dim $w_2$ plane. We have $O(N^2)$ switching lines, where each line generates only one intersection point with the projection of $\Delta^1$, leading to $O(N^2)$ intersection points.

Inductive step ($d = x + 1$). $\Delta^x$ can be projected onto a $x$-dim plane. We have $O(N^2)$ switching $x$-dim planes. Each two $x$-dim planes may intersect at an $(x - 1)$-dim plane, thus determining $O((N^2)^2)$ $(x-1)$-dim planes. By inductive hypothesis, $O(Y^2)$ $(x - 1)$-dim planes determine $O(Y^{2^{x-1}})$ intersection points, so, by letting $Y = N^2$, the number of possible intersection points is $O(N^{2^x})$. □

### 3.1.2 Computing MPO

We first consider the case of $d = 2$. We start with an initial ordering $\boldsymbol{\lambda}_N$ corresponding to weights $(1.0, 0)$, and we scan the set of switching planes in ascending order of $w_2$. We merge consecutive partitions of $\Delta^1$ that agree on the ordering of the top-$K$ join results (i.e., we merge any two consecutive partitions if their boundary switching plane does not change the rank of any join result in the top-$K$). We update ranks in $\boldsymbol{\lambda}_N$ at each switching plane. Eventually, the largest merged partition gives $\boldsymbol{\lambda}^*_{\text{MPO}}$. The complexity of the algorithm is dominated by sorting $O(N^2)$ switching planes, and hence the overall complexity is $O(N^2 \log N)$. We discuss in Section 3.3 an important optimization to significantly reduce $N$ in the previous complexity bound by pruning all $K$-dominated join results.

For example, in Figure 5(a), for $K = 2$ we have $\boldsymbol{\lambda}^*_{\text{MPO}} = \langle \tau_2, \tau_3 \rangle$, since it corresponds to the partition [1/6, 3/5], which is the largest partition inducing the same top-2 join results. Similarly, for $K = N$ we have $\boldsymbol{\lambda}^*_{\text{MPO}} = \langle \tau_2, \tau_3, \tau_4, \tau_1 \rangle$.

For $d > 2$, sorting the switching planes is not possible since, in contrast to the case of $d = 2$, the planes are not 1-dimensional anymore. Computing $\boldsymbol{\lambda}^*_{\text{MPO}}$ in this case is done by progressively partitioning the space using the switching planes, while maintaining the polyhedra corresponding to different top-$K$ answers. Thus, $\boldsymbol{\lambda}^*_{\text{MPO}}$ is given by the polyhedron with the largest volume (we elaborate on volume

computation in Section 3.2). Based on Theorem 3.2, the previous algorithm has complexity $O(N^{2^{d-1}})$.

### 3.1.3 Computing ORA

We first discuss computing $\boldsymbol{\lambda}^*_{\text{ORA}}$ for $d = 2$, and then present a generalization for $d > 2$ at the end of this section.

**ORA under Kendall tau Distance ($d = 2$).** We start by defining the property of Weak Stochastic Transitivity [15] in our settings.

DEFINITION 3.3. [**Weak Stochastic Transitivity**] $\mathcal{F}(\tau)$ *is weak stochastic transitive iff* $\forall \ \tau_i, \tau_j, \tau_k \in \mathcal{O}$: $[\alpha_{i,j} \geq 0.5 \ and \ \alpha_{j,k} \geq 0.5] \Rightarrow \alpha_{i,k} \geq 0.5.$ □

We show in Lemma 3.4 that weak stochastic transitivity always holds for $d = 2$.

LEMMA 3.4. $\mathcal{F}(\tau)$ *is weak stochastic transitive for* $d = 2$.

PROOF. Let $\tau_i, \tau_j, \tau_k$ be three join results, where $\alpha_{i,j} \geq 0.5$ and $\alpha_{j,k} \geq 0.5$. We show that we must also have $\alpha_{i,k} \geq 0.5$. Based on Lemma 3.1, we have $\alpha_{i,j} = p(w_2 > \frac{C_{i,j}}{1+C_{i,j}}) \geq 0.5$. As shown in Figure 4(a), the previous inequality holds if and only if $\frac{C_{i,j}}{1+C_{i,j}} \leq 0.5$, or equivalently, $C_{i,j} \leq 0.5 + 0.5 C_{i,j}$, which yields $C_{i,j} \leq 1.0$. By expanding $C_{i,j}$, we get the following expression:

$$s_{i,2} - s_{j,2} \geq s_{j,1} - s_{i,1} \tag{7}$$

It is easy to show that the opposite also holds. That is, (7) is a necessary and sufficient condition for $\alpha_{i,j} \geq 0.5$. Specifically, given $s_{i,2} - s_{j,2} \geq s_{j,1} - s_{i,1}$, we get $C_{i,j} \leq 1.0$, which yields $\frac{C_{i,j}}{1+C_{i,j}} \leq 0.5$, and hence $\alpha_{i,j} \geq 0.5$.

Similar to (7), using the given $\alpha_{j,k} \geq 0.5$, we get the following expression:

$$s_{j,2} - s_{k,2} \geq s_{k,1} - s_{j,1} \tag{8}$$

Adding up (7) and (8) gives $s_{i,2} - s_{k,2} \geq s_{k,1} - s_{i,1}$, which implies that $\alpha_{i,k} \geq 0.5$, and hence weak stochastic transitivity holds. □

As was argued in the proof of Lemma 3.4, (9) is a necessary and sufficient condition for $\alpha_{i,j} \geq 0.5$:

$$[\alpha_{i,j} \geq 0.5] \Leftrightarrow [s_{i,1} + s_{i,2} \geq s_{j,1} + s_{j,2}] \tag{9}$$

That is, if $\tau_i$ precedes $\tau_j$, in the order of scores' summation, it is guaranteed that $\alpha_{i,j} \geq 0.5$.

Assume that we would like to compute the total Kendall tau distance between an ordering $\bar{\boldsymbol{\lambda}}$ and all orderings in $\Lambda_N$. Based on Problem ORA, an ordering $\boldsymbol{\lambda} \in \Lambda_N$ contributes to the aggregated distance between $\bar{\boldsymbol{\lambda}}$ and $\Lambda_N$, with a value of $p(\boldsymbol{\lambda})$, for each pair of join results with disagreeing relative orders in $\bar{\boldsymbol{\lambda}}$ and $\boldsymbol{\lambda}$. Let $\Lambda_N^{i,j} \subseteq \Lambda_N$ be the set of possible orderings where $\tau_i$ is ranked before $\tau_j$. Then, $\alpha_{i,j} = \sum_{\boldsymbol{\lambda} \in \Lambda_N^{i,j}} p(\boldsymbol{\lambda})$, and it follows that the objective of Problem ORA under Kendall tau distance can be restated as finding an ordering that minimizes the total disagreements given by the following penalty function:

$$pen(\bar{\boldsymbol{\lambda}}) = \sum_{(\tau_i,\tau_j):\bar{\boldsymbol{\lambda}}(\tau_j)<\bar{\boldsymbol{\lambda}}(\tau_i)} \alpha_{i,j} \qquad (10)$$

**Construction Algorithm.** Assume an ordering $\bar{\boldsymbol{\lambda}}$ constructed as follows: For two distinct join results $\tau_i$ and $\tau_j$, let $\bar{\boldsymbol{\lambda}}(\tau_i) < \bar{\boldsymbol{\lambda}}(\tau_j)$ if $\alpha_{i,j} \geq 0.5$ while breaking probability ties deterministically. Based on weak stochastic transitivity, such construction procedure does not introduce cycles in $\bar{\boldsymbol{\lambda}}$. Moreover, we show that such construction procedure indeed minimizes $pen(.)$. Our result is the same as Theorem 3 in [13], and we include it here for completeness: ‡

THEOREM 3.5. *If weak stochastic transitivity holds, then an ordering $\bar{\boldsymbol{\lambda}}$, with $\bar{\boldsymbol{\lambda}}(\tau_i) < \bar{\boldsymbol{\lambda}}(\tau_j)$ if $\alpha_{i,j} \geq 0.5$, is $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under Kendall tau distance.* □

It follows that $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ is given by sorting join results using $(\alpha_{i,j} \geq 0.5)$ as the sort comparator. Based on (9), the sorting is achieved by ordering join results on $(s_{.,1} + s_{.,2})$. The total complexity is $O(NlogN)$, which is the complexity of conducting a regular sort.
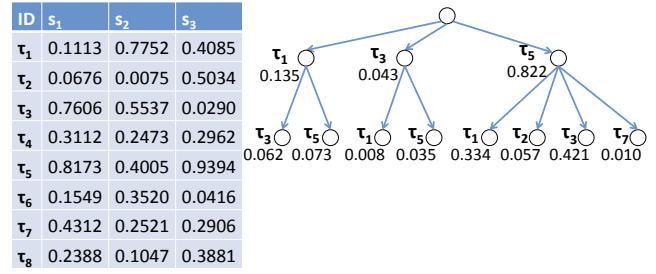
**ORA under Footrule Distance ($d = 2$).** Given an orderings space $\Lambda_N$, finding $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under footrule distance is studied in [4], where a weighted bipartite graph $G$ is used to connect each item $\tau$ with each rank $r$ using an edge weighted as $\sum_{\boldsymbol{\lambda}\in\Lambda_N} |\boldsymbol{\lambda}(\tau) - r|$, or equivalently (as shown in [13]), as $\sum_{i=1}^{N} p_{\tau,i} \times |i - r|$, where $p_{\tau,i}$ is the probability summation of orderings in $\Lambda_N$ with $\tau$ at rank $i$. Then, $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ is given by the minimum cost perfect matching of $G$, which can be found in $O(N^{2.5})$ [4].

We show how to construct the graph $G$ using our representation of $\Lambda_N$. We maintain for each $\tau$ a set $\{p_{\tau,i}\}$ of $N$ variables such that each variable $p_{\tau,i}$ represents the current probability of having $\tau$ at rank $i$. Starting from the initial ordering $\boldsymbol{\lambda}_N$ corresponding to weights $(1.0, 0)$, we scan the set of sorted switching planes in ascending order of $w_2$, while updating ranks in $\boldsymbol{\lambda}_N$ at each switching plane. Whenever we access a partition of $\Delta^1$ with $\tau$ at rank $i$, we add the length of that partition to the current $p_{\tau,i}$ value. The final $\{p_{\tau,i}\}$ values give the weights of edges incident to $\tau$ in $G$. For example in Figure 5(a), for $\tau_3$ we have $p_{\tau_3,1} = 0.4$, $p_{\tau_3,2} = 0.433$, $p_{\tau_3,3} = 0.167$, and $p_{\tau_3,4} = 0$. Since the number of switching planes is in $O(N^2)$, the complexity of constructing $G$ is $O(N^2 logN)$, and hence the overall complexity of finding $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ is dominated by the cost of perfect matching of $G$, which is $O(N^{2.5})$ based on [4].

**Computing ORA in Higher Dimensions.** Finding $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under Kendall tau distance is in general NP-Hard [4]. For $d = 2$, weak stochastic transitivity allows us to give an $O(NlogN)$ algorithm. However, weak stochastic transitivity does not hold for $d > 2$ †, and hence, $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ can only be approximated in polynomial time using, for example, the Markov chain sampling method given in [4]. We leave out the details in the interest of space.

---

‡The scoring model of [13] is different from ours. We prove that weak stochastic transitivity also holds under our scoring model, and hence $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ can be computed efficiently.
†A counterexample: For $\mathbf{s}(\tau_1) = [42, 0, 0]$, $\mathbf{s}(\tau_2) = [0, 40, 0]$, $\mathbf{s}(\tau_3) = [0, 28, 10]$, we have $\alpha_{1,2} = .51$, $\alpha_{2,3} = .54$, $\alpha_{1,3} = .48$.



| ID | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|
| $\tau_1$ | 0.1113 | 0.7752 | 0.4085 |
| $\tau_2$ | 0.0676 | 0.0075 | 0.5034 |
| $\tau_3$ | 0.7606 | 0.5537 | 0.0290 |
| $\tau_4$ | 0.3112 | 0.2473 | 0.2962 |
| $\tau_5$ | 0.8173 | 0.4005 | 0.9394 |
| $\tau_6$ | 0.1549 | 0.3520 | 0.0416 |
| $\tau_7$ | 0.4312 | 0.2521 | 0.2906 |
| $\tau_8$ | 0.2388 | 0.1047 | 0.3881 |

**Figure 6: Possible orderings tree.**

Computing $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under footrule distance for $d > 2$ can be done by computing the minimum cost perfect matching of the graph $G$, as discussed for the case of $d = 2$. However, the difference is that for $d > 2$, the cost of constructing $G$ dominates the cost of solving the matching problem (which is $O(N^{2.5})$), since $G$ is given by materializing the orderings space. Based on Theorem 3.2, the overall complexity of computing $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ is thus $O(N^{2^{d-1}})$. It is also known that $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under footrule distance is a 2-approximation of $\boldsymbol{\lambda}^*_{\mathrm{ORA}}$ under Kendall tau distance [4].

## 3.2 An Incremental Approach

We represent possible orderings as a tree, where each node at depth $k$ encodes a possible ordering prefix of size $k$, given by the path from the tree root to that node. A probability value is assigned to each node. We describe the details of tree construction in Section 3.2.1, while we discuss computing MPO in Section 3.2.2.

### 3.2.1 Possible Orderings Representation

For the sake of clarity, consider the problem of determining the possible top-1 answers and the 3-dimensional score vectors space depicted in Figure 6, where $N = 8$. By linearity of the aggregation function, we base our method on computing the convex hull of the score vectors $\mathbf{s}(\tau_i)$, $i = 1, \ldots, N$. The term "convex hull" usually refers to the boundary of the minimal convex set containing a set of vectors. In the following, when referring to the convex hull of the score vectors, we shall only retain those score vectors belonging to facets whose normal is directed towards the first orthant. This constraint stems from the fact that the weights are nonnegative, i.e., $w_i \geq 0$, $i = 1, \ldots, d$.

Intuitively, out of the $\binom{N}{1} 1! = N$ join results (i.e., sets of cardinality equal to 1), only the $M_1$ join results whose score vector belongs to the convex hull can be selected as top-1 answers, i.e., $\boldsymbol{\lambda}^1_1 = \langle \tau_1 \rangle$, $\boldsymbol{\lambda}^2_1 = \langle \tau_3 \rangle$ and $\boldsymbol{\lambda}^3_1 = \langle \tau_5 \rangle$. These join results are shown as nodes at depth $k = 1$ in the tree depicted in Figure 6. All the others are dominated by at least one score vector on the convex hull, regardless of the specific linear aggregation function. This observation has been previously made in [2], where *onion indexes* were designed with the goal of efficiently pre-computing the answers of top-$K$ queries. Note that, for uniformly distributed score vectors in $[0, 1]^3$, the number $M_1$ of score vectors on the convex hull increases asymptotically according to $O((\log N)^2)$ [5], thus it is typically much smaller than $N$.

In order to compute $p(\boldsymbol{\lambda}^r_1)$ we are interested in partitioning $\Delta^2$ in $M_1$ polygons $\pi(\boldsymbol{\lambda}^r_1)$, $r = 1, \ldots, M_1$, where each polygon corresponds to the set of weights for which a join result $\tau_j$ is selected as top-1, i.e., $\boldsymbol{\lambda}^r_1 = \langle \tau_j \rangle$. To this end, we consider the score vectors connected to $\mathbf{s}(\tau_j)$ on the convex hull, i.e., those score vectors $\mathbf{s}(\tau_{j_e})$, $e = 1, \ldots, E$, delimiting
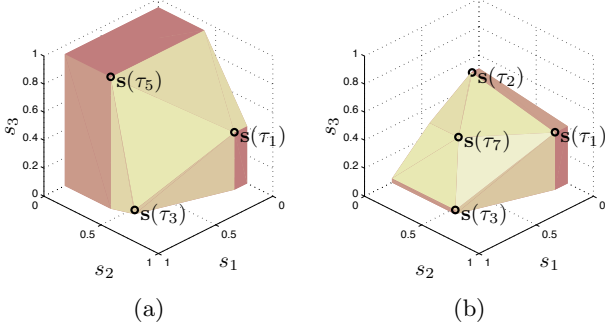
**Figure 7: (a) Convex hull for top-1 (b) Convex hull for top-2 when $\tau_5$ is top-1**
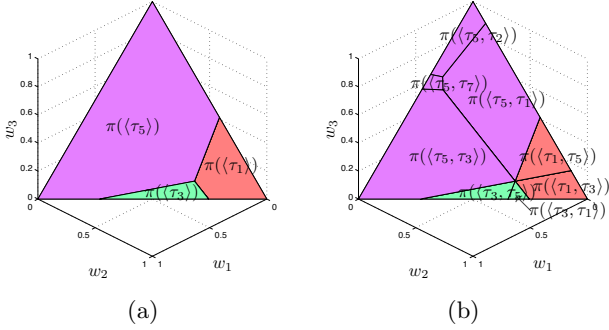


**Figure 8: Partitioning of $\Delta^2$ (a) at depth $K = 1$ (b) at depth $K = 2$**

edges departing from $\mathbf{s}(\tau_j)$, and impose that their aggregate score does not exceed that of $\tau_j$ by writing

$$
\begin{aligned}
\mathbf{w}^T \mathbf{s}(\tau_j) &\geq \mathbf{w}^T \mathbf{s}(\tau_{j_1}) \\
&\cdots \\
\mathbf{w}^T \mathbf{s}(\tau_j) &\geq \mathbf{w}^T \mathbf{s}(\tau_{j_E})
\end{aligned} \tag{11}
$$

together with the implicit constraint that $\mathbf{w}$ belongs to the simplex, i.e., $\mathbf{w} \in \Delta^2$. Thus, we can express $\pi(\boldsymbol{\lambda}_1^r) = \pi(\langle \tau_j \rangle)$ as the set $\{\mathbf{w} \in \Delta^2 | \mathbf{A}\mathbf{w} \leq \mathbf{b}, \mathbf{A} = \mathbf{A}(\boldsymbol{\lambda}_1^r), \mathbf{b} = \mathbf{b}(\boldsymbol{\lambda}_1^r)\}$, which defines a convex polygon. For instance, Figure 7(a) depicts the convex hull created using the possible top-1 join results. There are 3 join results on the hull surface, while other join results are inside the hull. Figure 8(a) illustrates the partitioning of $\Delta^2$ corresponding to the score vectors depicted in Figure 7(a).

In order to find the $M_2$ top-2 possible answers we proceed as follows. For each of the possible top-1 orderings $\boldsymbol{\lambda}_1^r$, $r = 1, \ldots, M_1$, we recompute the convex hull after removing the score vector associated to the join result in $\boldsymbol{\lambda}_1^r$. Let $\tau_j$ denote such a join result, i.e., $\boldsymbol{\lambda}_1^r = \langle \tau_j \rangle$, and $\tau_{j,l}$, $l = 1, \ldots, M_{1,j}$, denote the join results whose score vectors are on the updated convex hull. For instance, Figure 7(b) depicts the modified convex hull obtained after removing $\mathbf{s}(\tau_5)$, which includes $\mathbf{s}(\tau_1)$, $\mathbf{s}(\tau_2)$, $\mathbf{s}(\tau_3)$ and $\mathbf{s}(\tau_7)$. The candidate top-2 answers $\boldsymbol{\lambda}_2^r$ can be expressed as $\langle \tau_j, \tau_{j,l} \rangle$, $j \in \{i | \exists r. \boldsymbol{\lambda}_1^r = \langle \tau_i \rangle\}$, $l = 1, \ldots, M_{1,j}$. For each candidate, we are interested in determining the polygon $\pi(\boldsymbol{\lambda}_2^r) = \pi(\langle \tau_j, \tau_{j,l} \rangle)$, which can be expressed by means of a set of linear constraints

$$
\begin{aligned}
\mathbf{w}^T \mathbf{s}(\tau_j) &\geq \mathbf{w}^T \mathbf{s}(\tau_{j,l}) \\
\mathbf{w}^T \mathbf{s}(\tau_{j,l}) &\geq \mathbf{w}^T \mathbf{s}(\tau_{j,l_1}) \\
&\cdots \\
\mathbf{w}^T \mathbf{s}(\tau_{j,l}) &\geq \mathbf{w}^T \mathbf{s}(\tau_{j,l_E})
\end{aligned} \tag{12}
$$

---

**Algorithm 1:** buildTree($K$, $MPO$, $[\mathbf{s}(\tau_1), \ldots, \mathbf{s}(\tau_N)]$)

*// Input: result size $K$, boolean $MPO$, score vectors matrix $[\mathbf{s}(\tau_1), \ldots, \mathbf{s}(\tau_N)]$*
*// Output: top-K tree*
$\mathbf{S} := pruneDominatedObjects([\mathbf{s}(\tau_1), \ldots, \mathbf{s}(\tau_N)], K)$;
**if** ($MPO$) **then** $Tree :=$ addNodesMPO(empty tree, $\mathbf{S}$, $K$, 1, $\emptyset$, 0);
**else** $Tree :=$ addNodes(empty tree, $\mathbf{S}$, $K$, 1, $\emptyset$);
**return** $Tree$;

---

**Algorithm 2:** addNodes($Tree$, $\mathbf{S}$, $K$, $k$, $\boldsymbol{\lambda}$)

*// Input: current tree $Tree$, score vectors matrix $\mathbf{S}$, result size $K$,*
*//      current depth $k$, current prefix $\boldsymbol{\lambda}$*
*// Output: top-K tree*
**if** ($k \leq K$) **then**
  Remove from $\mathbf{S}$ all score vectors in $\boldsymbol{\lambda}$;
  Compute convex hull $\mathcal{H}$ of remaining vectors;
**for each** ($\tau_j \in \mathcal{H}$)
  Find objects connected to $\tau_j$;
  Compute linear constraints and $\pi(\boldsymbol{\lambda} \circ \tau_j)$;
  **if** ($\pi(\boldsymbol{\lambda} \circ \tau_j)$ is not empty) **then**
    Add $\tau_j$ to branch $\boldsymbol{\lambda}$ in $Tree$;
    $Tree :=$ addNodes($Tree$, $\mathbf{S}$, $K$, $k + 1$, $\boldsymbol{\lambda} \circ \tau_j$);
**return** $Tree$;

---

together with the implicit constraint $\mathbf{w} \in \Delta^2$, where $\mathbf{s}(\tau_{j,l_e})$, $e = 1, \ldots, E$, denotes the score vectors that belong to the convex hull obtained after removing $\mathbf{s}(\tau_j)$, which are connected to $\mathbf{s}(\tau_{j,l})$. For example, when $\langle \tau_5 \rangle$ is top-1, the following candidate top-2 answers can be obtained: $\boldsymbol{\lambda}_2^1 = \langle \tau_5, \tau_1 \rangle$, $\boldsymbol{\lambda}_2^2 = \langle \tau_5, \tau_2 \rangle$, $\boldsymbol{\lambda}_2^3 = \langle \tau_5, \tau_3 \rangle$ and $\boldsymbol{\lambda}_2^4 = \langle \tau_5, \tau_7 \rangle$. After that, computing, say, $\pi(\langle \tau_5, \tau_3 \rangle)$ requires to consider $\mathbf{s}(\tau_1)$ and $\mathbf{s}(\tau_7)$, since they are directly connected to $\mathbf{s}(\tau_3)$. Note that, for some candidate top-2 answers, the linear system of equations in (12) might not have a feasible solution and $\pi(\langle \tau_j, \tau_{j,l} \rangle) = \emptyset$. This is due to the fact that there is no linear aggregation function for which the corresponding candidate answers can be selected as top-2. Thus, such candidates are pruned from the set of possible orderings.

In the example given in Figure 6, the number of possible top-2 answers equals the number of leaves in the tree. Note that, out of $\binom{N}{K}K! = \binom{8}{2}2! = 56$ orderings, only 8 of them are identified as possible top-2 answers.

The determination of top-$K$ possible answers proceeds similarly, by recursively applying the same step individually to each leaf node of the top-$(K-1)$ tree.

Algorithm 1 illustrates the procedure for constructing the tree of possible top-$K$ answers in detail. $Tree$ represents the tree topology, whereas $\boldsymbol{\lambda}$ indicates the current branch of the tree, i.e., an input ordering for which the first $k-1$ elements have been computed. We also call *depth* the length of the prefix of $\boldsymbol{\lambda}$ computed by the algorithm. After an initial pruning of all dominated objects (discussed in Section 3.3), the algorithm recursively calls *addNodes* (or *addNodesMPO*, if the tree is used to compute MPO) in a depth-first fashion, until the desired depth $K$ is reached. In the *addNodes* and *addNodesMPO* functions, reported in Algorithms 2 and 3, resp., we indicate as $\circ$ the concatenation of an element to a sequence (namely, of a join result to a prefix of an ordering).

The average time complexity of constructing the top-$K$ tree can be derived for the case of uniformly distributed score vectors. For $K \ll N$, each node has $O((\log N)^2)$ children. Therefore, there are $O((\log N)^{2K})$ non-leaf nodes for which the convex hull needs to be computed. For $d = 3$, specialized algorithms exist that compute the convex hull with

---
**Algorithm 3:** addNodesMPO($Tree$, $\mathbf{S}$, $K$, $k$, $\boldsymbol{\lambda}$, $p_{max}$)
---
// Input: current tree $Tree$, score vectors matrix $\mathbf{S}$, result size $K$,
//        current depth $k$, current prefix $\boldsymbol{\lambda}$, current MPO probability $p_{max}$
// Output: (partial) top-$K$ tree
**if** $(k \leq K)$ **then**
  Remove from $\mathbf{S}$ all score vectors in $\boldsymbol{\lambda}$;
  Compute convex hull $\mathcal{H}$ of remaining vectors;
**for each** $(\tau_j \in \mathcal{H})$
  Find objects connected to $\tau_j$;
  Compute linear constraints and $\pi(\boldsymbol{\lambda} \circ \tau_j)$;
  Compute $p(\boldsymbol{\lambda} \circ \tau_j)$;
**for each** $(\tau_j \in \mathcal{H}$ in descending order of $p(\boldsymbol{\lambda} \circ \tau_j))$
  **if** $(p(\boldsymbol{\lambda} \circ \tau_j) > p_{max})$ **then**
    Add $\tau_j$ to branch $\boldsymbol{\lambda}$ in $Tree$;
    **if** $(k = K)$ **then** $p_{max} := p(\boldsymbol{\lambda} \circ \tau_j)$;
    **else** $Tree$ := addNodesMPO($Tree$, $\mathbf{S}$, $K$, $k+1$, $\boldsymbol{\lambda} \circ \tau_j$, $p_{max}$);
**return** $Tree$;
---

complexity $O(N \log(N))$ [3]. Hence, the overall asymptotic complexity is $O(N(\log N)^{2K+1})$.

We discuss in Section 3.3 how to significantly reduce $N$ in the previous bound by pruning all $K$-dominated join results.

**Computing Nodes Probabilities.** When the weight vector $\mathbf{w}$ is uniformly distributed on $\Delta^2$, evaluating $p(\boldsymbol{\lambda})$ requires computing the area of the polygon $\pi(\boldsymbol{\lambda})$. This can be done in three steps: *i)* Compute the coordinates of the vertices of the polygon on $\Delta^2$, i.e., $[w_1^v, w_2^v]^T$, $v = 1, \ldots, V$. The vertex enumeration problem can be solved in $O(CV)$ time [1], where $C$ denotes the number of linear constraints and $V$ the number of vertices. The resulting asymptotic complexity is dominated by the top-$K$ tree construction, since the number of constraints is $O((\log(N))^2)$ and there might be up to $O((\log(N))^4)$ intersections (candidate vertices). *ii)* Compute the area by means of the Shoelace formula

$$A = \frac{1}{2} \left| \sum_{v=1}^{V} w_1^v w_2^{v+1} - \sum_{v=1}^{V} w_1^{v+1} w_2^v \right| \quad (13)$$

where $w_i^{V+1} = w_i^1$. *iii)* $p(\boldsymbol{\lambda}) = 2A$.

For example, Figure 6 illustrates, for each set of possible orderings $\Lambda_1$ and $\Lambda_2$, the corresponding probabilities.

**Handling Higher Dimensions.** The method proposed for $d = 3$ continues to be applicable for larger values of $d$, as far as the enumeration of possible orderings is concerned, although the asymptotic complexity of computing the convex hull is $O(N^{\lfloor d/2 \rfloor + 1})$ instead of $O(N \log(N))$ [3]. Unfortunately, determining the probabilities $p(\boldsymbol{\lambda})$ requires computing the volume of convex polyhedra embedded in $\Delta^{d-1}$, which is NP-hard. It is still possible to determine a lower and upper bound on $p(\boldsymbol{\lambda})$ in polynomial time, by computing, respectively, the maximum volume inscribed ellipsoid and the minimum volume enclosing ellipsoid.

In order to find an approximation of $p(\boldsymbol{\lambda})$, we propose a Monte-Carlo sampling method. For a space of orderings $\Lambda_K = \{\boldsymbol{\lambda}_K^1, \ldots, \boldsymbol{\lambda}_K^m\}$, we maintain a counter $c_r$ for each ordering $\boldsymbol{\lambda}_K^r \in \Lambda_K$. We sample the space of weights uniformly at random. For each sample weight vector $\bar{\mathbf{w}}$, where $\mathcal{O} \overset{\bar{\mathbf{w}}}{\rightsquigarrow} \boldsymbol{\lambda}_K^r$, we increment the counter of $\boldsymbol{\lambda}_K^r$ by 1. Let the number of drawn samples be $s$. The value of $p(\boldsymbol{\lambda}_K^r)$ is approximated as $c_r/s$. The approximation error is in $O(1/\sqrt{s})$, which is the error of Monte-Carlo sampling.

### 3.2.2  Computing MPO

A naïve method to compute $\boldsymbol{\lambda}_{\text{MPO}}^*$ consists of the following steps: *i)* construct the (full) top-$K$ tree; *ii)* compute the probability of each leaf node $p(\boldsymbol{\lambda}_K)$; *iii)* select the node associated with the largest probability. We notice that, in order to speed up the computation of $\boldsymbol{\lambda}_{\text{MPO}}^*$, some of the branches of the top-$K$ tree can be safely pruned without the need of reaching all leaf nodes. To this end, we interleave the tree construction with the computation of the probabilities of visited nodes (both intermediate and leaf nodes) and keep track of the current MPO candidate among the visited leaf nodes. We observe that each node is assigned a probability which cannot be greater than the probability of its parent. Therefore, it is possible to safely prune those branches rooted at a node whose probability is less than that of the current MPO candidate.

The amount of pruning depends on the order followed to traverse the tree. Ideally, the algorithm should reach the leaf node corresponding to $\boldsymbol{\lambda}_{\text{MPO}}^*$ as early as possible in such a way that most of the nodes can be pruned. Therefore, at each level of the tree, nodes are explored in descending order of probability. This behavior is implemented in Algorithm 3.

## 3.3  Pruning $K$-Dominated Join Results

When computing MPO at depth $K$, an important optimization is to start by pruning all $K$-dominated join results. A join result $\tau_i$ is $K$-dominated if there exists a set of $\geq K$ join results whose score vectors dominate the score vector of $\tau_i$. A $K$-dominated join result does not appear in any possible top-$K$ answer, and hence it can be safely pruned without affecting the correctness of the computed MPO. Since $K$ is typically a small number, the effect of $K$-dominance on reducing the number of join results is substantial.

A naïve algorithm to prune $K$-dominated join results is to count, for each join result $\tau_i$, the number of other join results dominating $\tau_i$, and prune $\tau_i$ as soon as such count is $K$. The complexity of this algorithm is $O(N^2)$. We show that we can achieve the same goal with more optimized algorithms.

For $d = 2$, we adopt the DominatingSet algorithm in [14], which has complexity $O(N log N)$. Extending this algorithm to handle $d > 2$ requires maintaining intersections of sets of join results, which reduces to $O(N^2)$ complexity. We thus adopt a different algorithm, based on Fagin's Algorithm (FA) [6], which, in practice, prunes the majority of $K$-dominated join results when $d > 2$.

First, each join result $\tau$ is assigned a unique identifier. Then, we create $d$ sorted lists $L_1, \ldots, L_d$, where each $L_i$ maintains join results' identifiers in the order of $s_{.,i}$. The cost of this step is $O(dN \log N)$. Let $\tau_i^{(x)}$ denote the identifier of the join result at position $x$ in $L_i$. The sorted lists $L_1 \ldots, L_d$ are scanned in parallel until a depth $\bar{x}$ is reached such that at least $K$ common identifiers are found among the $d$ lists of seen identifiers. That is, $|\{\tau_1^{(1)}, \ldots, \tau_1^{(\bar{x})}\} \cap \ldots \cap \{\tau_d^{(1)}, \ldots, \tau_d^{(\bar{x})}\}| \geq K$. Let $U$ denote the union of the seen identifiers, i.e., $U = \{\tau_1^{(1)}, \ldots, \tau_1^{(\bar{x})}\} \cup \ldots \cup \{\tau_d^{(1)}, \ldots, \tau_d^{(\bar{x})}\}$. The top-$K$ join results are guaranteed to be among those whose identifiers are in the set $U$, for any monotonic aggregation function. Therefore, those join results whose identifiers are not contained in $U$ can be safely pruned. It is shown in [6] that, if the rankings are independent and over the same set of $N$ items, the cost for finding the top-$K$ an-

swer is $O(N^{\frac{d-1}{d}}K^{\frac{1}{d}})$, which is sub-linear in $N$. Hence, the cost of pruning is dominated by the initial sort operation.

## 4. WEIGHT PREFERENCES

We discuss handling user's preferences specified in the form of a strict partial order on the weights. A strict partial order on the weights is a binary preference relation $\succ_{\mathcal{P}}$ that is irreflexive (i.e., $(w_i \not\succ_{\mathcal{P}} w_i)$), asymmetric (i.e., $(w_i \succ_{\mathcal{P}} w_j) \Rightarrow (w_j \not\succ_{\mathcal{P}} w_i)$), and transitive (i.e., $[(w_i \succ_{\mathcal{P}} w_j), (w_j \succ_{\mathcal{P}} w_k)] \Rightarrow (w_i \succ_{\mathcal{P}} w_k)$). For the example in Figure 2, a user interested in join results with highly-rated restaurants, may express such preference using a partial order $(w_R \succ_{\mathcal{P}} w_H)$. Note that for $d = 2$, a partial order on the weights is effectively a total order, while for $d > 2$, a partial order can be a non-total order (e.g., $(w_1 \succ_{\mathcal{P}} w_2), (w_1 \succ_{\mathcal{P}} w_3)$).

Each preference $(w_i \succ_{\mathcal{P}} w_j)$ implies a constraint $(w_i > w_j)$ on the weights space. Let $W_{\mathcal{P}} = \{\mathbf{w} \in \Delta^{d-1} | (w_i > w_j) \; whenever \; (w_i \succ_{\mathcal{P}} w_j)\}$. Based on the properties of strict partial orders, $W_{\mathcal{P}}$ is expressed in terms of linear constraints, and hence $W_{\mathcal{P}}$ is a convex polyhedron embedded in $\Delta^{d-1}$. We define a uniform probability density function over $W_{\mathcal{P}}$ as follows:

$$p(\mathbf{w}) = \begin{cases} \beta & \text{if } \mathbf{w} \in W_{\mathcal{P}} \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

where $\beta$ is a normalizing constant such that the integral of $p(\mathbf{w})$ over $W_{\mathcal{P}}$ is equal to one.

The techniques presented in Sections 3.1 and 3.2 remain applicable to this scenario. To illustrate, assume that $d = 2$. Weight preferences give a total order on $\{w_1, w_2\}$, which implies that one of the two weights is greater than 0.5 (since $w_1 + w_2 = 1$). Hence, $W_{\mathcal{P}}$ is given by either the upper half of $\Delta^1$ (if $w_2 > w_1$), or the lower half of $\Delta^1$ (if $w_1 > w_2$). Probabilities are thus given based on partitions of only one half of $\Delta^1$. For example, in Figure 5(a), $\alpha_{3,2} = 0$ if $w_1 > w_2$, since in this case only the lower half of $\Delta^1$ is considered.

## 5. SENSITIVITY MEASURES

When a user provides an explicit weight vector, a corresponding ordering is immediately obtained. The user may be interested in receiving additional feedback indicating the largest perturbation in the input weights that does not affect the ordering (Problem STB) as well as the likelihood of the ordering (Problem LIK). Such feedback can be important in interactive data analysis, and guiding scoring function tuning to capture user's preferences. We discuss our proposed solutions to these problems next.

### 5.1 Stability of the Ordering wrt. the Weights

Let $\bar{\mathbf{w}}$ be a weight vector, where $\mathcal{O} \stackrel{\bar{\mathbf{w}}}{\leadsto} \bar{\boldsymbol{\lambda}}_N$. Let $\pi(\bar{\boldsymbol{\lambda}}_K)$ be the polyhedron defining the set of weights inducing $\bar{\boldsymbol{\lambda}}_K$. We show how to solve Problem STB, where we find the radius $\rho_K(\bar{\mathbf{w}})$ of the maximum volume hypersphere $\sigma_K(\bar{\mathbf{w}})$ centered at $\bar{\mathbf{w}}$, and enclosed in $\pi(\bar{\boldsymbol{\lambda}}_K)$.

Computing $\rho_K(\bar{\mathbf{w}})$ requires determining the distances between $\bar{\mathbf{w}}$ and each of the hyperplanes delimiting the polyhedron $\pi(\bar{\boldsymbol{\lambda}}_K)$. There are up to $N-1$ such hyperplanes, defined as follows:

$$\mathbf{w}^T \mathbf{s}(\tau_{(1)}) \geq \mathbf{w}^T \mathbf{s}(\tau_{(2)})$$
$$\mathbf{w}^T \mathbf{s}(\tau_{(2)}) \geq \mathbf{w}^T \mathbf{s}(\tau_{(3)})$$
$$\cdots \geq \cdots$$
$$\mathbf{w}^T \mathbf{s}(\tau_{(K-1)}) \geq \mathbf{w}^T \mathbf{s}(\tau_{(K)})$$
$$\mathbf{w}^T \mathbf{s}(\tau_{(K)}) \geq \mathbf{w}^T \mathbf{s}(\tau_{(K+1)})$$
$$\cdots \geq \cdots$$
$$\mathbf{w}^T \mathbf{s}(\tau_{(K)}) \geq \mathbf{w}^T \mathbf{s}(\tau_{(N)}) \quad (15)$$

where $\tau_{(i)}$ is the $i$-th combination in $\bar{\boldsymbol{\lambda}}_N$.

Since there are $O(N)$ hyperplanes, and computing the distance between a point and a plan in a $d$-dimensional space can be done in $O(d)$, the worst case time complexity for computing $\rho_K(\bar{\mathbf{w}})$ is $O(dN)$. The cost can be significantly reduced to be sublinear in $N$ if the score vectors have been pre-indexed, e.g., removing dominated vectors or computing the tree of possible orderings, as illustrated in Section 3.2.1.

Alternatively, one can measure stability of $\bar{\mathbf{w}}$ by aggregating all the radii $\rho_k(\bar{\mathbf{w}})$ for $k = 1, \ldots, K$ into a single value:

$$\tilde{\rho}_K(\bar{\mathbf{w}}) = \frac{1}{K\rho_{\max}} \sum_{k=1}^{K} \rho_k(\bar{\mathbf{w}}) D(k) \quad (16)$$

where $i)$ $\rho_{\max}$ is the radius of the maximum volume hypersphere enclosed in $\Delta^{d-1}$, and $ii)$ $D(k)$ is a discount function adopted to emphasize changes that affect the top elements in the ordering, e.g., $D(k) = 1/\log(2 + k)$.

### 5.2 Ordering Likelihood

Given a weight vector $\bar{\mathbf{w}}$, where $\mathcal{O} \stackrel{\bar{\mathbf{w}}}{\leadsto} \bar{\boldsymbol{\lambda}}_N$, let $\bar{\boldsymbol{\lambda}}_N = \langle \tau_{(1)}, \tau_{(2)}, \ldots \tau_{(N)} \rangle$. Let $\pi(\bar{\boldsymbol{\lambda}}_K)$ be the polyhedron defining the set of weights inducing $\bar{\boldsymbol{\lambda}}_K$. Then, $\gamma_K(\bar{\boldsymbol{\lambda}}_N)$ is given by the ratio of the volume of $\pi(\bar{\boldsymbol{\lambda}}_K)$ and the volume of $\Delta^{d-1}$. We define $\pi(\bar{\boldsymbol{\lambda}}_K)$ similar to Section 5.1: We compute the switching planes of $(\tau_{(i)}, \tau_{(i+1)})$ for $i = 1 \ldots K - 1$, and $(\tau_{(K)}, \tau_{(j)})$ for $K < j \leq N$. The number of these switching planes is in $O(N)$, and $\pi(\bar{\boldsymbol{\lambda}}_K)$ is the minimal convex polyhedron, created by the intersections of these switching planes, enclosing $\bar{\mathbf{w}}$.

For $d = 2$, $\pi(\bar{\boldsymbol{\lambda}}_K)$ is given by a partition of the $w_2$ axis. This allows computing $\gamma_K(\bar{\boldsymbol{\lambda}}_N)$ in $O(N)$ by finding the closest switching plane above $\bar{\mathbf{w}}$, and the closest switching plane below $\bar{\mathbf{w}}$, on the $w_2$ axis. $\pi(\bar{\boldsymbol{\lambda}}_K)$ is the partition enclosed between these two closest planes. For example, in Figure 5(a), for $\bar{\mathbf{w}} = (0.6, 0.4)$, we have $\bar{\boldsymbol{\lambda}}_4 = \langle \tau_2, \tau_3, \tau_4, \tau_1 \rangle$. Then, if $K = 2$, we need to compute the switching planes of $(\tau_2, \tau_3)$, $(\tau_3, \tau_4)$, and $(\tau_3, \tau_1)$. The closest top and bottom planes wrt. $\bar{\mathbf{w}} = (0.6, 0.4)$ are the planes of $(\tau_2, \tau_3)$ and $(\tau_3, \tau_4)$, respectively. Hence, $\pi(\bar{\boldsymbol{\lambda}}_K)$ is given by the partition `[1/6,3/5]`, which leads to $\gamma_2(\bar{\boldsymbol{\lambda}}_4) = 0.43$.

For $d = 3$, the number of intersection points of $O(N)$ switching planes is in $O(N^2)$. The vertices of $\pi(\bar{\boldsymbol{\lambda}}_K)$ are among these intersection points. Hence, $\pi(\bar{\boldsymbol{\lambda}}_K)$ can be found in $O(N^2)$ by inspecting all possible polygons to find the minimal convex polygon enclosing $\bar{\mathbf{w}}$. For example, in Figure 5(a), for $\bar{\mathbf{w}} = (0.5, 0.4)$, we have $\bar{\boldsymbol{\lambda}}_4 = \langle \tau_1, \tau_2, \tau_3, \tau_4 \rangle$. Then, if $K = 2$, $\pi(\bar{\boldsymbol{\lambda}}_K)$ is given by the polygon `[(0.21,0.3)-(0.6,0.4)-(0.14,0.86)]` whose area can be computed as shown in (13), giving $\gamma_2(\bar{\boldsymbol{\lambda}}_4) = 0.2254$.

In general, for $d$ dimensions, the complexity of computing Problem LIK is $O(N^{2^{d-2}})$, which is the bound on the num-

**Table 1: CPU time to compute MPO on synthetics data sets, Zipfian distribution**

| | $d=2$ | | | $d=3$ | | | $d=4$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $N$ | $K=1$ | $K=5$ | $K=10$ | $K=1$ | $K=5$ | $K=10$ | $K=1$ | $K=5$ | $K=10$ |
| 100 | $1.3 \cdot 10^{-3}$ | $9.8 \cdot 10^{-4}$ | $8.8 \cdot 10^{-3}$ | $1.1 \cdot 10^{-5}$ | $6.2 \cdot 10^{-5}$ | $2.0 \cdot 10^{-4}$ | $2.0 \cdot 10^{-3}$ | $1.6 \cdot 10^{-2}$ | $2.5 \cdot 10^{-1}$ |
| 1000 | $1.6 \cdot 10^{-4}$ | $1.6 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $1.6 \cdot 10^{-5}$ | $1.6 \cdot 10^{-4}$ | $1.1 \cdot 10^{-3}$ | $1.7 \cdot 10^{-2}$ | $3.2 \cdot 10^{-2}$ | $2.0 \cdot 10^{-1}$ |
| 10000 | $4.6 \cdot 10^{-4}$ | $4.6 \cdot 10^{-3}$ | $2.8 \cdot 10^{-2}$ | $6.2 \cdot 10^{-5}$ | $5.9 \cdot 10^{-4}$ | $5.9 \cdot 10^{-3}$ | $1.0 \cdot 10^{-1}$ | $1.5 \cdot 10^{-1}$ | $3.3 \cdot 10^{-1}$ |
| 100000 | $3.3 \cdot 10^{-3}$ | $2.9 \cdot 10^{-2}$ | $1.5 \cdot 10^{-1}$ | $6.4 \cdot 10^{-4}$ | $1.7 \cdot 10^{-3}$ | $2.1 \cdot 10^{-2}$ | $7.1 \cdot 10^{-1}$ | $1.2 \cdot 10^{0}$ | $1.6 \cdot 10^{0}$ |

**Table 2: CPU time to compute MPO on synthetics data sets, exponential distribution**

| | $d=2$ | | | $d=3$ | | | $d=4$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $N$ | $K=1$ | $K=5$ | $K=10$ | $K=1$ | $K=5$ | $K=10$ | $K=1$ | $K=5$ | $K=10$ |
| 100 | $1.6 \cdot 10^{-4}$ | $3.3 \cdot 10^{-3}$ | $2.2 \cdot 10^{-2}$ | $1.1 \cdot 10^{-5}$ | $4.7 \cdot 10^{-5}$ | $2.2 \cdot 10^{-4}$ | $2.3 \cdot 10^{-3}$ | $3.4 \cdot 10^{-2}$ | $3.8 \cdot 10^{-1}$ |
| 1000 | $2.6 \cdot 10^{-4}$ | $4.9 \cdot 10^{-3}$ | $3.3 \cdot 10^{-2}$ | $1.1 \cdot 10^{-5}$ | $2.2 \cdot 10^{-4}$ | $1.3 \cdot 10^{-3}$ | $1.8 \cdot 10^{-2}$ | $6.1 \cdot 10^{-2}$ | $4.4 \cdot 10^{-1}$ |
| 10000 | $7.1 \cdot 10^{-4}$ | $1.1 \cdot 10^{-2}$ | $7.0 \cdot 10^{-2}$ | $3.1 \cdot 10^{-5}$ | $9.8 \cdot 10^{-4}$ | $9.3 \cdot 10^{-3}$ | $1.2 \cdot 10^{-1}$ | $2.0 \cdot 10^{-1}$ | $5.8 \cdot 10^{-1}$ |
| 100000 | $3.9 \cdot 10^{-3}$ | $6.8 \cdot 10^{-2}$ | $4.2 \cdot 10^{-1}$ | $6.2 \cdot 10^{-4}$ | $2.9 \cdot 10^{-3}$ | $2.2 \cdot 10^{-2}$ | $7.1 \cdot 10^{-1}$ | $1.0 \cdot 10^{0}$ | $1.6 \cdot 10^{0}$ |



**Figure 12: Relative approximation error ($d=4$)**

| Data set | $d=2$ | $d=3$ | $d=4$ |
|---|---|---|---|
| New York | 3871 | 41907 | 209599 |
| San Francisco | 9016 | 56200 | 591952 |
| Boston | 4712 | 9740 | 39030 |
| Dallas | 1424 | 1806 | 15196 |

**Table 3: Number $N$ of join results for real data sets**

ber of minimal convex polyhedra created by the intersections of $O(N)$ switching planes. However, for $d > 3$, computing a polyhedron's volume is NP-Hard, and hence we need to approximate $\gamma_K(\bar{\boldsymbol{\lambda}}_N)$ as discussed in Section 3.2.1.

We conclude this section with one last result on the likelihood of $\boldsymbol{\lambda}^*_{\text{ORA}}$ under Kendall tau distance and $d = 2$.

Let $w_2^+$ be the first intersection point, between a switching plane and the $w_2$ axis, above 0.5. When there is no such point, let $w_2^+ = 1.0$. Similarly, let $w_2^-$ be the first intersection point, between a switching plane and the $w_2$ axis, below 0.5. When there is no such point, let $w_2^- = 0$. Then, $\boldsymbol{\lambda}^*_{\text{ORA}}$ is represented by the $\Delta^1$ partition defined in Theorem 5.1:

THEOREM 5.1. *When there is no $\alpha_{i,j} = 0.5$, $\boldsymbol{\lambda}^*_{ORA}$ under Kendall tau distance is given by the partition $[w_2^-, w_2^+]$. Otherwise, $\boldsymbol{\lambda}^*_{ORA}$ is given by either the partition $[0.5, w_2^+]$, or the partition $[w_2^-, 0.5]$.*

PROOF. When there is no $\alpha_{i,j} = 0.5$, then for all $\alpha_{i,j} < 0.5$, the switching plane of $(\tau_i, \tau_j)$ is located above (or at) $w_2^+$, while for all $\alpha_{i,j} > 0.5$, the switching plane of $(\tau_i, \tau_j)$ is located below (or at) $w_2^-$. Hence, for all $(\tau_i, \tau_j)$, with $\alpha_{i,j} > 0.5$, the ordering induced by the partition $[w_2^-, w_2^+]$ positions $\tau_i$ above $\tau_j$. This means that $[w_2^-, w_2^+]$ induces $\boldsymbol{\lambda}^*_{\text{ORA}}$ according to Theorem 3.5.

Similarly, when there is at least one $\alpha_{i,j} = 0.5$, any one of the two partitions $[0.5, w_2^+]$, or $[w_2^-, 0.5]$ induces an ordering that positions $\tau_i$ above $\tau_j$ iff $\alpha_{i,j} \geq 0.5$, which means that either partition induces $\boldsymbol{\lambda}^*_{\text{ORA}}$. □
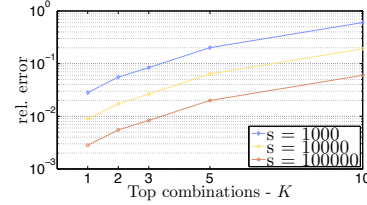
For the example in Figure 5(a), $\boldsymbol{\lambda}^*_{\text{ORA}} = \langle \tau_2, \tau_3, \tau_1, \tau_4 \rangle$ and $\gamma_4(\boldsymbol{\lambda}^*_{\text{ORA}}) = 0.17$, which is the length of the partition [3/7, 3/5].

# 6. EXPERIMENTS

In this section we experimentally evaluate the algorithms described in this paper and study the impact of various parameters on execution: number of desired results, number of relations, number of join results, and data distribution.

## 6.1 Methodology

**Data sets.** First, we conduct our analysis on synthetic data sets. We generated $d$ independent relations of tuples with scores sampled from a selected distribution among uniform, Zipfian, and exponential, such that their join produces $N$ join results, with $2 \leq d \leq 4$ and $10^2 \leq N \leq 10^5$. While the number of join results can be large, our study involves a parameter $K$ whose value is typically small. This leads

to aggressive pruning of the join results (cf. Section 3.3). Hence, the subset of join results we need to process is usually small, even though the number of all join results is large.

Then, we consider real data sets with relations containing hotels, restaurants, theaters, and cars for rent in four different American cities (New York, San Francisco, Boston, and Dallas). Each data set is obtained by retrieving customer ratings, latitude and longitude of each such place by means of the YQL console[1]. For each city, the relations are joined in such a way that only the join results in which the mutual distance between the objects is below a certain threshold are retained. Table 3 shows the number of join results $N$ computed for each data set when $d$ relations are joined.

**Methods and evaluation metrics.** We test the different algorithms described in this paper to compute MPO. For $d = 2, 3$ the result is exact, in the sense that the output of the algorithm is the correct ordering $\boldsymbol{\lambda}^*_{\text{MPO}}$ together with its probability $p(\boldsymbol{\lambda}^*_{\text{MPO}})$. For $d = 4$ we resort to sampling to determine the possible orderings and their probabilities. As such, the result is an approximation of the correct MPO.

*Total CPU time*, in seconds, is the metric we adopt for evaluating our results. The total CPU time also includes the time needed to prune dominated join results, as described in Section 3.3. For fairness, in every experiment we run our tests over ten different data sets and report the average.

*Relative approximation error* is the metric we adopt for evaluating the accuracy of the solution produced by sampling. The relative approximation error is given by $|\hat{p}(\boldsymbol{\lambda}^{\hat{*}}_{\text{MPO}}) - p(\boldsymbol{\lambda}^*_{\text{MPO}})|/p(\boldsymbol{\lambda}^*_{\text{MPO}})$, where $\hat{p}(\boldsymbol{\lambda}^{\hat{*}}_{\text{MPO}})$ is the approximated value obtained by sampling.

*Number of children* is the metric we adopt for characterizing the structure of the tree constructed in the incremental approach, which gives an indication of the number of possible orderings of length $K$.

We do not run experiments to test ORA for the following reasons. For $d = 2$ (under Kendall tau distance) the solution amounts to sorting, so testing would uninterestingly evaluate the efficiency of a sorting algorithm. For $d > 2$, ORA is NP-hard; approximations of the Kendall tau distance are possible, and they are described in other works [4].

Determining the largest perturbation that can be tolerated without affecting the ordering, as described in Section 5.1, is at most linear in the number of join results. Hence, the total CPU time needed is negligible when compared to the other problems addressed in this paper.

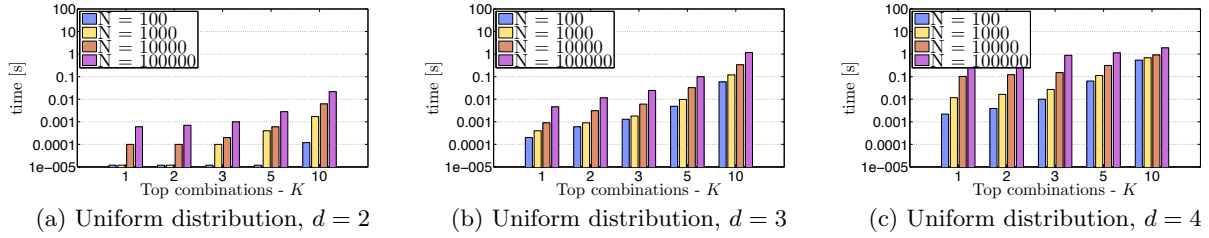Computing ordering likelihood has polynomial time com-

---

[1] Available at `http://developer.yahoo.com/yql/console/`

(a) Uniform distribution, $d = 2$     (b) Uniform distribution, $d = 3$     (c) Uniform distribution, $d = 4$

**Figure 9: CPU time to compute MPO vs. number of top results on synthetics data sets**



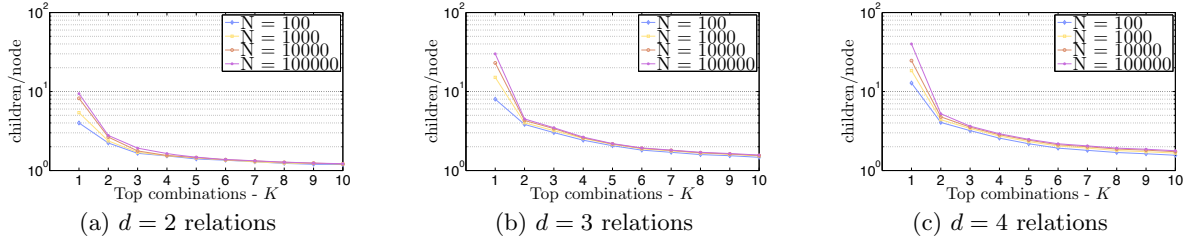(a) $d = 2$ relations     (b) $d = 3$ relations     (c) $d = 4$ relations

**Figure 10: Number of children per child node vs. number of top results**

plexity in $N$, as shown in Section 5.2. We do not include corresponding experimental results due to space constraints.

**Testing environment** All the tests have been run on a PC with the Windows 7 operating system, an Intel® Core2-Duo processor at 2.4GHz and 4Gb RAM. We measured the wall clock execution time needed to compute MPO for a given $K$, assuming that the join results are available locally. Thus, we did not consider the time needed for fetching the tuples when data come from remote sources.

## 6.2 Results

The results obtained on the synthetic data sets are summarized in Figure 9, Table 1 and Table 2. The bars in the charts represent the total CPU time needed to compute MPO for a given $K$ (between 1 and 10), and include the time needed for the initial pruning. The average total CPU time in our experiments is always less than 10 seconds, and exceeding 1 second only for very large values of $N$ (100000), which is perfectly reasonable for the scenarios at stake.

All our experiments include dominance-based pruning of join results, since we have observed that this determines remarkable improvements in terms of total CPU time.

We now comment on the individual parameters.

**Number of relations** - $d$: For $d = 2$, Figure 9(a) reports the time required to compute MPO after using an exact pruning for the case of a uniform distribution. The reported times are in the worst case around .02 seconds.

For $d = 3$, Figure 9(b) reports the time required to compute MPO after using an approximate pruning step that discards most of the $K$-dominated join results (cf. Section 3.3). Here, we adopted Algorithm 3, which partially constructs the possible orderings tree, pruning those branches that cannot contain the MPO. For comparison, Figure 13 shows the total CPU time needed to compute the full tree according to Algorithm 2, comprising all possible orderings, with the same score distribution. The two algorithms are identical for $K = 1$, whereas for larger values of $K$ the speed up due to Algorithm 3 can be as large as three orders of magnitude.

For $d = 4$, we compute MPO by sampling possible orderings and count how many times they occur. The total CPU time depends linearly on the number of samples $s$; the dependency on $K$ is only due to pruning, while the depen-
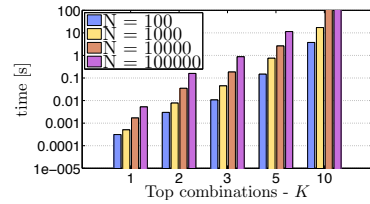


**Figure 13: CPU time vs. number of top results on synthetics data sets for $d = 3$ (full tree construction)**

dency on $N$ is due to both pruning and sorting. We also assess the relative approximation error introduced by sampling in Figure 12, by averaging over values of $N$. With $s = 1000$ samples, the error grows by a factor of $\sqrt{10}$, while the total CPU time would scale down by a factor of 10.

**Number of join results** - $N$: Increasing $N$ determines visibly higher total CPU times in all experiments, as $N$ affects all algorithms (for any $d$). Yet, even when $N$ is large, the total CPU time is always within an acceptable range.

**Score distribution** - We tested three different score distributions: uniform, Zipfian and exponential. There is a remarkable difference between different score distributions when computing the trees of possible orderings if no pruning is applied: the fan-out is of the order of $O((\log N)^{d-1})$ if a uniform distribution is adopted, while it is much smaller with the other distributions. However, after pruning, this gap is smoothed out, and no significant difference is visible in the results, as reported in Tables 1 and 2.

For the incremental approach, we also evaluated the structure of the tree in terms of the average number of children per node (a.k.a. fan-out) at different depths, as illustrated in Figure 10 for the case of uniformly distributed scores. We observe that the fan-out decreases with depth, approaching the asymptotic value of 1 child per node. At a given depth $K$, the fan-out increases with $d$. For example, at $K = 1$, the average fan-out is $O((\log(N))^{d-1})$. Note that the average number of possible orderings of length $K$ is readily obtained by multiplying the values of fan-out between 1 and $K$.

Finally, Figure 11 shows our results for real data sets. In San Francisco we found the largest number of join results (nearly 600000 for $d = 4$), thus with a total CPU time higher
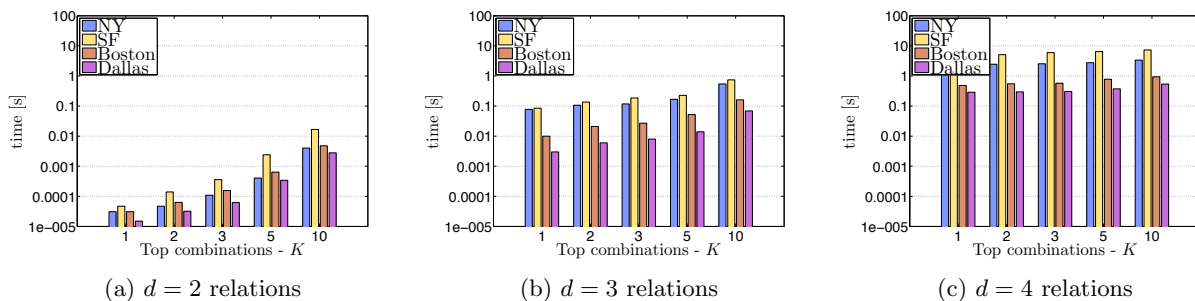
(a) $d = 2$ relations       (b) $d = 3$ relations       (c) $d = 4$ relations

**Figure 11: CPU time to compute MPO vs. number of top results on real data sets**

than in other cities. Approximate pruning is still beneficial (the number drops down to 200000 for $K = 10$), whereas exact pruning would be too costly and has been avoided.

## 7. RELATED WORK

The problem of rank join has been addressed in many studies [7, 8, 11]. The central idea in most proposals is to compute the top-$K$ join results while limiting data access and join operations based on scoring function monotonicity and sorted access of joined relations. This paper extends rank join methods by handling uncertain scoring functions with partially-specified weight preferences.

The methods in [10, 12, 13] assume uncertain continuous scores defined as independent random variables. With uncertain scoring functions, tuples' scores become correlated with scoring function instances, and hence models and ranking techniques assuming independence do not apply.

The closest proposals to our study are top-$K$ indexing techniques [2, 14]. In general, indexing top-$K$ answers, for a family of scoring functions, assumes pre-defined joined relations/scoring attributes and no selection conditions, which may limit index usability in interactive data analysis. Our techniques do not make such assumptions, since answers are efficiently computed on the fly. We elaborate on other differences with these proposals in the following.

Onion Indices [2] are used to index a set of $d$-dimensional objects, where each dimension is a scoring attribute. The boundary of the smallest convex hull that encloses all objects is guaranteed to include the top-1 object (assuming a linear scoring function). Onion Indices extend this observation by constructing layered convex hulls to index objects for efficient top-$K$ processing. We build on similar observations for building the possible orderings tree (cf. Section 3.2.1).

Ranked Join Indices [14] materialize possible top-$K$ answers based on weighted summation of two scoring attributes. The proposed methods in [14] only handle $d = 2$, while our methods handle $d \geq 2$. Moreover, while the scenario in [14] involves equally-likely orderings, we offer a novel probabilistic interpretation of possible orderings (originating from score distributions and weight preferences) to handle the scenario where a user formulates preferences as uncertain scoring functions. The notions of MPO and ORA (cf. Section 2.2) have not been addressed before in this scenario.

Rank aggregation of uncertain data has been addressed in [9] under a model that decouples data uncertainty from scoring measures. Under our model, however, uncertainty is induced by a user-defined scoring measure.

## 8. CONCLUSIONS

We study the semantics and sensitivity of ranking query answers for scoring functions with uncertain weights. In particular, we focus on formulating representative orderings and computing sensitivity measures that can be given to the user as a feedback under weights uncertainty. We present efficient algorithms to solve these problems, and show that we handle a larger number of scoring dimensions compared to current proposals.

## 9. ACKNOWLEDGEMENT

## 10. REFERENCES

[1] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8, 1992.

[2] 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. In *SIGMOD*, 2000.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.

[4] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

[5] R. A. Dwyer and R. Kannan. Convex hull of randomly chosen points from a polytope. In *Parallel Algorithms and Architectures*, 1987.

[6] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2), 2002.

[7] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, 2003.

[8] 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(4), 2008.

[9] J. Li and A. Deshpande. Consensus answers for queries over probabilistic databases. In *PODS*, 2009.

[10] J. Li and A. Deshpande. Ranking continuous probabilistic datasets. *PVLDB*, 3(1), 2010.

[11] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, 2001.

[12] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, 2009.

[13] M. A. Soliman, I. F. Ilyas, and S. Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDB Journal*, 19(4), 2010.

[14] P. Tsaparas, N. Koudas, Y. Kotidis, T. Palpanas, and D. Srivastava. Ranked join indices. In *In ICDE*, 2003.

[15] P. van Acker. Transitivity revisited. *Ann. Oper. Res.*, 23(1-4), 1990.

[16] H. Yu, S. won Hwang, and K. C.-C. Chang. Enabling ad-hoc ranking for data retrieval. In *ICDE*, 2005.