Adaptive Synthetic Generation of Indefinite Rankings

Enhancing Algorithm Flexibility with Tunable Conjointness, Overlap, and Tie Distribution

Somdutta Sinha

Supervisor(s): Dr Julián Urbano, Matteo Corsi

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Somdutta Sinha
Final project course: CSE3000 Research Project
Thesis committee: Dr Julián Urbano, Matteo Corsi, Mathijs Molenaar

Abstract

Reducing the similarity of two ranked lists to a single value proves to be useful in various fields of research and industry, such as Information Retrieval and Recommender Systems, leading to the introduction of several similarity measures. One such measure is Rank-Biased Overlap and its variants, possessing qualities such as the ability to handle incomplete rankings, non-conjoint ranking domains and the graceful evaluation of ranking pairs with tied items. Comparing the performance of similarity measures, or comparing variants of a single measure, requires the presence of ranking data. In certain cases, generating synthetic ranking data may be a more viable option than using real data. However, a review of existing literature reveals a lack of parametrisable synthetic ranking algorithms. This paper introduces a novel method to generate a pair of rankings where one can tailor the conjointness of ranking domains, influence the ranking overlap as a function of depth and tune the presence of tie groups in a probabilistic manner. The paper demonstrates the output of the algorithm when varying the input parameters, verifying the methods performance empirically and statistically.

Keywords

Rankings, rank-biased overlap, synthetic ranking generation

1 Introduction

In the field of information retrieval, ranking algorithms play an important role in presenting information that is most relevant to users. These algorithms are found extensively in search engines and recommender systems where items from some domain need to be ranked. Objectively evaluating such algorithms often involves comparing pairs of ranked items to quantify the similarity of ranked lists, motivating the need for robust and reliable rank similarity measures [5].

Webber et al. introduced Rank-Biased Overlap (RBO) as a rank similarity measure with certain defining properties [5], distinguishing it from other traditional similarity metrics such as Pearson's correlation and Kendall's τ [3]. These properties include the ability to handle top-weighted and incomplete ranking pairs (see Section 2.1 for formal definitions). Furthermore, RBO can offer a monotonically increasing lower bound and monotonically decreasing upper bound of the final score when evaluated on prefixes of the rankings.

Despite their widespread use in information retrieval - Sharma et al. describe and review twelve distinct page ranking algorithms [4] - the existing method for generating synthetic rankings to test RBO and other rank similarity measures have certain limitations, particularly in the lack of flexibility to tune certain parameters of the ranking such as the conjointness of the domains and the presence of distributions of ties.

In this paper, we consider the problem of synthetic ranking generation and improving the current method proposed by Corsi and Urbano in [1], while taking RBO's unique properties into account. This paper will propose a new method that allows researchers to tune various properties of rankings. Henceforth, the R script `src/simulate.R` in the GitHub repository `julian-urbano/sigir2024-rbo` [1] will be referred to as the 'current synthetic ranking generation method'.

In designing a new synthetic ranking generation method, the main research question - *the current method to simulate synthetic rankings is not tailored to RBO's properties. How can we adapt this simulation taking inspiration from RBO?* - will be addressed. To further break this down, the following sub-questions will be directly answered - upon which the basis of the improved ranking simulation algorithm can be established.

- What are the limitations of the current synthetic ranking generation method?
- How can the conjointness of the ranking domains be tuned?
- How can the overlap of items in rankings be adjusted given a probability function?
- How can the location and distribution of ties in a ranking be varied?

In Section 2, the research background will be introduced, firstly formalising the RBO measure and secondly providing an overview of the existing synthetic ranking generation method, identifying limitations. Section 3 will underline the methodology adopted to answer the sub-questions raised and thus establish the skeleton of the final algorithm. The final rank simulation algorithm will be formally defined and explained in Section 3.5. The performance of the algorithm will be presented in various forms in Section 4. The reader will clearly be able to see the effects of tuning input hyper-parameters to different values. Section 5 will note the ethical considerations and responsible research measures undertaken. The results presented in Section 4 will be thoroughly evaluated and discussed in Section 6. Finally, Section 7 will conclude the research undertaken and offer points of improvement and future research.

To facilitate reproducibility and further research, the algorithm developed in this paper is available on GitHub [2].

2 Background

2.1 Rank-Biased Overlap

Many existing measures view similarity between ranked lists as a question of bi-variate correlation. Although in certain, constrained scenarios these measures may be appropriate, it is often the case that ranked lists do not satisfy these constraints. For example, it may be the case that the two rankings do not come from conjoint domains. Furthermore, Webber et al. note that researchers comparing ranked lists may also desire a measure with the following properties [5]:

- top-weightedness: ranking overlap at earlier depths should be favoured over overlap at later depths.

[1] https://github.com/julian-urbano/sigir2024-rbo

[2] https://github.com/somduttasinha/synthetic-ranking-generation

- incompleteness: it may be the case that only a strict subset of the entire domain is ranked, while the rest of the domain remains un-ranked. The two subsets may also be disjoint.
- indefiniteness: in certain cases where the domain is so large such that the calculation of a measure based on exhaustively evaluating the entire domain is computationally expensive, an evaluator may want the ability to place a bound on the final value of the evaluation.

Webber et al. define a ranking with these properties as an 'indefinite ranking'. They demonstrate that no existing rank correlation technique can handle all three properties at the same time.

The property of top-weightedness is addressed through a parameter p, indicating the *persistence* of the evaluator [5]. A larger value of p means that items at lower ranks are given more weight than when a smaller value of p is used. If $p = 1$, all items are given the same weight, if $p = 0$, only the first item is considered from each ranking. The persistence parameter p must be between 0 and 1.

To address the issue of non-conjointness, rather than considering rank similarity as a correlation, RBO is a direct extension of set overlap [5], where the overlap of two ranked lists, S and L, at some depth d is the intersection of all items observed in set S until d and all items in set L until d.

RBO uses the concept of 'prefix-evaluation' to achieve bounds in the case of indefinite rankings. Given two prefixes, a minimum bound RBO_{MIN} and a maximum bound RBO_{MAX} can be computed by assuming that either the entire unseen part is disjoint or fully conjoint respectively. Furthermore, a point estimate RBO_{EXT} can be found by assuming that the observed agreement, $\frac{X_d}{d}$, in the seen part will not change for the rest of the ranking. The rank-biased overlap of two lists S and L is given by:

$$RBO(p) = \frac{1-p}{p} \sum_{d=1}^{\infty} \frac{X_d}{d} p^d \tag{1}$$

where X_d is the size of the overlap between two ranked lists S and L until depth d [5].

Webber et al. also consider the cases where the prefixes are of different lenths and the presence of ties in the rankings. This work is expanded by Corsi and Urbano in [1].

2.2 Current Synthetic Ranking Generation Algorithm

Before introducing the simulation algorithm in the following section, we briefly explore the existing algorithm introduced by Corsi and Urbano in [1] and identify limitations in this approach.

Corsi and Urbano's approach works by first simulating a pair of fully conjoint rankings. Ties are then introduced into both rankings before being truncated to the user's specified length.

Simulating Conjoint Rankings

The 'similarity' of the rankings is governed by the `tau` parameter. A pair of rankings is synthesised by first simulating two correlated vectors coming from a Gaussian copula where the covariance is a function of `tau` [1].

Introducing Ties and Truncating Rankings

Ties are introduced into the ranking in the function `make_ties` by grouping items into tied ranks based on the number of tie groups. This function takes a vector of scores, `x`, desired number of tied items, `n_ties` and number of tie groups, `n_groups` as input parameters. Firstly, the vector `[1, 1, 2, 2, ..., n_groups, n_groups]` is shuffled. This vector corresponds to the group label that the input vector scores will be assigned to. This vector is of size $2 * $ `n_groups` - this can be understood intuitively as at minimum, each tie group should contain at least two items by definition. If `n_ties` is greater than $2 * $ `n_groups`, the group label vector needs to be extended. This is done by sampling from the vector `[1, 2, ..., n_groups]`. However this time, the sampling is not done uniformly as equally-sized tie groups are not realistic; instead, the vector is sampled according to a probability vector that is generated from a Dirichlet($\boldsymbol{q}$) distribution where $\boldsymbol{q}$ is a `n_groups`-length vector with each entry q_i sampled from a $U(0, 1)$ distribution.

This vector is extended by adding labels for each unique untied item. Finally, `make_ties` allocates each score from the input vector `x` into their assigned group.

The simulation algorithm finishes by assigning each score a unique ID in the `score2id` function and truncating each generated ranking to the desired length if required. A user looking to introduce some level of disjointness to the rankings can do so by truncating the generated rankings to `len_x` and `len_y` while using scores of size `n`. Naturally, a large difference between the truncated lengths and the length of the simulated scores, n, will lead to greater disjointness.

Algorithm Parameters

Corsi and Urbano's algorithm allows the user to input the following parameters:

- `len_x` and `len_y`
- `enforce_ties`
- `n`
- `tau`
- `frac_ties_x` and `frac_ties_y`
- `n_groups_x` and `n_groups_y`

If `enforce_ties` is set to `TRUE`, the algorithm will repeat until it is able to generate a pair of rankings which both have ties. Furthermore, if a parameter is not provided, it is assigned a random value.

Limitations

Although this algorithm is capable of tuning certain parameters such as the desired correlation of the rankings, we immediately notice that there is a lack of fine-grained control over other aspects of the rankings. For example, the user cannot precisely control the nature of ranking overlap as the depth increases, nor can they dictate the distribution of ties. Additionally, the method's reliance on random sampling and distribution parameters can lead to variability in the generated

rankings, potentially making it challenging to replicate specific ranking structures or achieve desired ranking characteristics consistently.

3 Constructing the Algorithm

The three key properties of ranking pairs identified were 1) conjointness of ranking domains, 2) the nature of the overlap between pairs of rankings and 3) the presence of ties in rankings. In this section, we consider each of these properties and incorporate the ability to tune them in the final ranking simulation algorithm.

Broadly, the algorithm takes the following shape:

1. Generate the domains for the rankings S and L to be drawn from using `a` and `b`, the size of each of the two domains and `jaccard_similarity` to influence the degree of conjointness of the two domains.
2. Create the two rankings, S and L by iteratively sampling from the domains created in (1). Sampling is not necessarily uniformly random, but is instead influenced by the input overlap probability function.
3. Introduce ties into the ranking by artificially selecting sequences of items following the tie distribution functions specified in the input.

3.1 Conjointness

One advantage of Rank-Biased Overlap as a rank similarity measure over other measures such as Kendall's τ [3] and Yilmaz et al.'s variant τ_{AP} [6] is the ability to handle disjoint domains [5]. The conjointness of two sets can be quanitfied concretely using the Jaccard similarity index [2]. This is calculated as a function J over two sets A and B.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

To allow for the conjointness of the domains to be tuned, the algorithm allows the user to input the desired `jaccard_similarity` as well as the size of the domains `a` and `b`. By re-arranging (2), we can find the cardinality of the required intersection and subsequently create two domains A and B that have the desired Jaccard similarity. Let the sizes of the two domains, $|A|$ and $|B|$ be a and b respectively:

$$\begin{aligned} |A \cup B| &= a + b - |A \cap B| \\ J(A, B)(a + b - |A \cap B|) &= |A \cap B| \\ J(A, B)(a + b) - J(A, B)(|A \cap B|) &= |A \cap B| \\ J(A, B)(a + b) &= J(A, B)(|A \cap B|) + |A \cap B| \end{aligned}$$

$$|A \cap B| = \frac{J(A, B)(a + b)}{1 + J(A, B)} \tag{3}$$

After finding the size of the intersection - we call this j, we can generate three sets:

1. $P = A \cap B$: prefix the character 'i' to all integers from 0 to $j - 1$ inclusive such that we obtain the set {'i0', 'i1', ..., 'i($j - 1$)'}
2. $Q = A \setminus P$: prefix the character 'a' to all integers from 0 to $a - j - 1$ inclusive such that we obtain the set {'a0', 'a1', ..., 'a($a - j - 1$)'}
3. $R = B \setminus P$: prefix the character 'b' to all integers from 0 to $b - j - 1$ inclusive such that we obtain the set {'b0', 'b1', ..., 'b($b - j - 1$)'}

From these sets, we can form the domains, A and B from which the rankings S and L will be drawn from respectively:

$$A = Q \cup P$$

$$B = R \cup P$$

3.2 Ranking Overlap

In Webber et al.'s paper introducing Rank-Biased Overlap, the idea of 'top-weightedness' was discussed where the authors assert that "the top of the list is more important than the tail" [5]. As such, the RBO formula (1) rewards overlaps at earlier depths as summands are weighted more than summands at later depths. From this, we see that an interesting case emerges where a ranking pair has agreements and overlap weighted towards the start of the ranking. RBO should evaluate the similarity of this ranking pair comparatively higher than a ranking pair where corresponding items are dispersed throughout the ranking.

For other purposes, it may be useful for a ranking pair to have a different overlap structure. For example, a researcher looking to compare the robustness of different rank similarity metrics when similarities are not observed until later rankings can define a probability function which is biased towards later depths (e.g. a scaled positive exponential function).

To accommodate for this, the algorithm allows the user to define a function f that takes as input the depth d and the domain size n. This function defines the *probability of increasing overlap*. The function f should be defined over all depths in the range $[0, n)$[3]. Furthermore, the range of f must be $[0, 1]$ as this constitutes a probability of increasing overlap.

In the algorithm at depth d, when sampling the sets A and B to find the next elements to add to the rankings S and L at position d, the function f is evaluated, yielding the probability $f(d) = \alpha$. With probability α, we force an increase in the overlap between S and L.

Formally, at each depth d, we sample from the random variable G_d, where $G_d \sim \text{Bernoulli}(\alpha)$. The outcome of a realisation of G_d determines the sampling strategy at d.

The overlap of two sets S and L can increase after choosing new items s and l if one of the following cases occur:

	s	l
I	item from L so far	randomly sampled
II	randomly sampled	item from S so far
III	random item r	same item r
IV	item from L so far	item from S so far

Table 1: Four possible cases to increase ranking overlap

[3]Rankings are typically 1-indexed while the function can take input values starting from 0. In the source code, the function is thus evaluated at $d - 1$ for a given depth d

We formalise these four cases using conventional set notation:

I. $s \in (A \setminus S_{:(d-1)}) \cap (L_{:(d-1)})$

II. $l \in (B \setminus L_{:(d-1)}) \cap (S_{:(d-1)})$

III. $s = l = r \in (A \setminus S_{:(d-1)}) \cap (B \setminus L_{:(d-1)})$

IV. $s \in (A \setminus S_{:(d-1)}) \cap L_{:(d-1)}$ and $l \in (B \setminus L_{:(d-1)}) \cap S_{:(d-1)}$

The probability of selecting each case is equal. This may lead to an 'unlucky' situation where one of these four cases is requested, but the set to sample from is empty. For example, if case I is drawn at the very start of the ranking where $d = 1$, $L_{:0}$ is empty by definition so nothing can be drawn. In this case, an item is drawn randomly from the ranking's domain.

Certain default functions are provided in `src/standard_functions.py`. These include the exponential decay function, the scaled logarithmic function[4] and the scaled standard normal probability density function[5]. If these functions were not scaled, it is possible that the probability output would never be large enough for the algorithm to choose to increase overlap. Thus, to mitigate this issue while maintaining the 'shape' of the function, each function f is scaled by $\frac{1}{\max_{d \in [0, n-1]} f(d)}$. From this, we see that for some depth d' for which f is maximum, the algorithm will force an increase in overlap with probability 1.

Uniformly Random Rankings

Before introducing the exponential decay function, it is worth considering the case where no overlap function is provided. In this case, the rankings generated are simply random permutations of the items in the given domain.

For conciseness, we will adopt the same notation used in [5] and [1], presented in Table 2.

S, L	Complete ranking as an ordered list
$S_{:d}$	A slice of S until and including the item at depth d
X_d	Overlap size of S and L at depth d
A_d	$\frac{X_d}{d}$, the agreement at depth d

Table 2: Notation used in this paper

For some S and L, drawn as independent, random permutations from some arbitrary n-sized domain D, we wish to observe the behaviour of $\mathbb{E}[A_d]$ as d increases to n. The probability that some arbitrary item x is in $S_{:d}$ is $\frac{d}{n}$. The same is true for L.

$$\mathbb{P}(x \in S_{:d}) = \mathbb{P}(x \in L_{:d}) = \frac{d}{n} \tag{4}$$

As items are placed in S and L independently, we find that the probability that x is in both $S_{:d}$ and $L_{:d}$ is given by:

$$\mathbb{P}(x \in S_{:d} \ \ \& \ \ x \in L_{:d}) = \left(\frac{d}{n}\right)^2 \tag{5}$$

[4] $f(d; n) = \frac{\ln(d)}{\ln(n+1)}$

[5] $f(d; \mu, \sigma) = \frac{\phi(d; \mu, \sigma)}{\phi(\mu; \mu, \sigma)}$

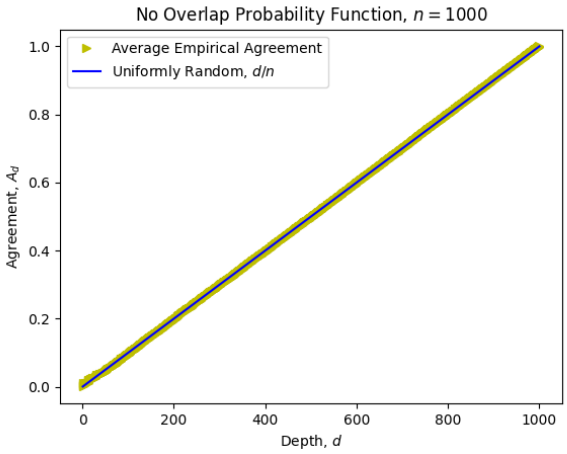

Figure 1: Plot of average agreement A_d as d increases to the domain size 1000 over 100 simulations. For reference, the linear function $\frac{d}{n}$ is also shown.

From (5), we can find the expected size of the overlap X_d:

$$\mathbb{E}[X_d] = \sum_{x \in D} \left(\frac{d}{n}\right)^2 = n\left(\frac{d^2}{n^2}\right) = \frac{d^2}{n} \tag{6}$$

Finally, we can find an expression for the expected agreement at depth d, A_d:

$$\mathbb{E}[A_d] = \frac{\mathbb{E}[X_d]}{d} = \frac{d}{n} \tag{7}$$

To verify this empirically, we run the simulation algorithm 100 times with no overlap probability function f proved, and plot the average agreement observed at each depth in Figure 1. For simplicity, we also choose to sample S and L from fully conjoint domain of size 1000. We see that the hypothesised expected agreement from Equation 7 is verified empirically.

Exponential decay

The user may want the rankings to have an alternative agreements profile. As discussed earlier, it may be interesting for researchers to simulate rankings where similarities between pairs of rankings are observed at earlier depths. Intuitively, this leads to the idea of a decaying overlap probability function f of the form $f(d) \propto e^{-d}$. Moreover, it is obvious that the function should scale with the size of the domain n - it should 'spread out' to span the entirety of the domain. Thus we include a term k that should be a function of the domain's size n. With this, we have:

$$f(d; n) = e^{-\frac{d}{k}}$$

$$k = g(n)$$

The user has freedom in choosing an appropriate function g. For example, one line of reasoning may decide that at some fraction t of the entire ranking, the likelihood of increasing overlap $f(tn)$ should be some fixed probability ω.

To illustrate this, let $t = \frac{1}{4}$ and $\omega = 0.3$. We have $f\left(\frac{1}{4}n\right) = 0.3$.

$$f\left(\frac{1}{4}n\right) = e^{-\frac{n}{4k}} = 0.3$$
$$\frac{-n}{4k} = \ln(0.3)$$
$$k = \frac{-n}{4\ln(0.3)} \tag{8}$$

Furthermore, we can parametrise this function to give the user further flexibility with the *degree* of top-weightedness. This is done by scaling the output of the exponential term by a scale term θ where $\theta \in [0, 1]$. This means that a user looking for a high-degree of top-weighted agreements will use an exponential function f with a value of θ close to 1. The final form of the overlap probability equation is given in (9), where k is given in (8).

$$f(d; n) = \theta e^{\frac{-n}{k}} \tag{9}$$

In Section 4, we demonstrate that increasing θ leads to an increase in RBO.

Analytical evaluation of the overlap function

A researcher developing their own overlap probability function may want to first predict the behaviour of its impact on the rankings' agreement with varying depth. To do this, this section will develop an analytical method to approximate expected ranking agreement as a function of depth. For simplicity, we assume that both S and L rank items from the same domain D.

After observing an overlap X_d at depth d, we see that the overlap at $d + 1$ increases with probability $f(d)$. However, it is also possible that we draw an item randomly that causes an increase in overlap. Let the expected increase in overlap with these two cases be denoted by t_1 and t_2 respectively. Then, we can find the expected overlap at $d + 1$:

$$\mathbb{E}[X_{d+1}] = \mathbb{E}[X_d] + f(d)(t_1) + (1 - f(d))(t_2) \tag{10}$$

- t_1: expected increase in overlap if we choose to increase overlap 'artificially'
- t_2: expected increase in overlap if items are drawn randomly

The expected increase in overlap if items are drawn randomly, t_2, is the probability of drawing an element for S from $L_{:d} - S_{:d}$ added to the probability of drawing an element for L from $S_{:d} - L_{:d}$. It is trivial to show that these two quantities are equal. At depth d, the total number of items from which S can draw an item from is $n - d$. It follows that the probability of drawing an item from $L_{:d} - S_{:d}$ is $\frac{1}{n-d}\mathbb{E}\left[|L_{:d} - S_{:d}|\right]$. To find $\mathbb{E}\left[|L_{:d} - S_{:d}|\right]$. we consider the probability that some arbitrary item x is in $L_{:d}$ and in $L_{:d} \cap S_{:d}$.

$$\mathbb{P}(x \in L_{:d}) = \frac{d}{n}$$
$$\mathbb{P}(x \in (L_{:d} \cap S_{:d})) = \frac{d^2}{n^2}$$
$$\mathbb{P}(x \in (L_{:d} - S_{:d})) = \mathbb{P}(x \in L_{:d}) - \mathbb{P}(x \in (L_{:d} \cap S_{:d}))$$
$$\mathbb{P}(x \in (L_{:d} - S_{:d})) = \frac{d}{n} - \frac{d^2}{n^2}$$
$$\mathbb{P}(x \in (L_{:d} - S_{:d})) = \frac{d}{n}\left(\frac{n-d}{n}\right) \tag{11}$$

To find the expected size of $L_{:d} - S_{:d}$, we multiply (11) by the size of the domain n. This gives us:

$$\mathbb{E}[|L_{:d} - S_{:d}|] = d\left(\frac{n-d}{n}\right) \tag{12}$$

From (12), we find that the expected increase in overlap as a cause of drawing an item from L for S is $\frac{d}{n}$. This is the same as the expected increase in overlap by drawing an item from S for L. Overall, we find that the expected increase in overlap if items are drawn randomly, t_2 from (10), is $\frac{2d}{n}$.

$$t_2 = \frac{2d}{n} \tag{13}$$

From Section 3.2, four ways to increase overlap were identified. Each case is chosen with equal probability. The expected increase for each case is:

I. $1 + \frac{d}{n}$: we assume that the set $(A - S_{:d}) \cap (L_{:d})$ has at least one element. $\frac{d}{n}$ is the expected increase in overlap by randomly selecting an item for L.

II. $\frac{d}{n} + 1$: same as I, except we choose randomly for S and increase overlap directly by choosing the item for L.

III. $1 - e^{-n(1-\frac{d}{n})^2}$: this is the probability that the set $(A - S_{:d}) \cap (B - L_{:d})$ is non-empty. To do this, we first find the expected size, γ, of the set $\mathbb{E}[|(A - S_{:d}) \cap (B - L_{:d})|]$. We then assume that the size of the set follows a Poisson(γ) distribution. The probability that the set is non-empty is calculated as $1 - \mathbb{P}$(set is empty).

IV. 2: we assume that there exists at least one element from $S_{:d}$ and $L_{:d}$ to add to the other ranking.

From this, we find that

$$t_1 = \frac{1}{4}\left(1 + \frac{d}{n}\right) + \frac{1}{4}\left(1 + \frac{d}{n}\right) + \frac{1}{4}\left(1 - e^{-n(1-\frac{d}{n})^2}\right) + \frac{1}{4}(2) \tag{14}$$

From (10), (13) and (14), the user can predict the shape of their chosen agreement probability function. Because we assume that each set from which we sample from contains at least one item, we see that the analytical estimate of the agreement will always be a strict overestimate of the true empirically observed agreement. An example of this assumption being invalid was explained earlier, where if case I was drawn at the $d = 1$, the set to sample from, $S_{:0} \cap L_{:0}$, was empty.

3.3 Introducing Ties

Ties are introduced to the ranking after two 'dummy' rankings are created, S and L, which exhibit the desired conjointness and overlap properties, but do not yet contain ties, in the `add_ties` function.

The `add_ties` function function takes as input a ranking `X` of size `n`, the fraction of items that should be tied, `frac_ties`, number of tie groups `num_groups` and the probabilities vector `probabilities`. Once the two rankings are created, `num_groups` indices are selected from the vector `[0, 1, ..., n-1]`, sampling according to the probability vector `tie_probabilities` using the NumPy function `np.random.choice(...)`[6]. These indices determine the starting indices of tie groups. This method ensures that exactly `num_groups` tie groups are generated. As a tie group must contain at least two items, the selected start indices vector is adjusted if necessary to ensure that there is a gap of at least 2 between successive indices.

The average tie group size is λ:

$$\lambda = \frac{\texttt{frac\_ties} * \texttt{n}}{\texttt{n\_groups}} \tag{15}$$

The size of a tie group is sampled from the random variable T, where:

$$T \sim Po(\lambda - 2) + 2 \tag{16}$$

to ensure that a sampled tie group size is always at least 2.

In a given ranking X, we add items to tie groups by iterating through each depth d. If d matches a selected start index, we sample T and assign it to `j`. We start a tie group and add all items from X_d to X_{d+j-1} inclusive.

On expectation, the number of tied items is $\mathbb{E}[T] *$ `n_groups`. From (16), we see that $\mathbb{E}[T] = \lambda$. By rearranging (15), we find that on average, the algorithm will tie `frac_ties` $*$ `n` items.

3.4 Input Parameters

- `a` and `b`: the length of the domains (and thus the full ranking).
- `len_x` and `len_y`: if requested, the lengths that the full rankings should be truncated to.
- `jaccard_similarity`: the desired Jaccard similarity of the domains.
- `overlap_probability_function`: a function $f(d)$ that associates each depth with a probability of increasing overlap.
- `tie_probabilities_x` and `tie_probabilities_y`: two vectors where each entry corresponds to the probability of starting a tie group at that index.
- `frac_ties_x` and `frac_ties_y`: the desired fraction of tied items.
- `n_groups_x` and `n_groups_y`: the desired number of tie groups.

[6] https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html

3.5 Algorithm

1. Generate two domains, A and B, using `jaccard_similarity` to determine the degree of conjointness.
2. Initialise two empty vectors, `S` and `L`.
3. Iteratively populate `S` and `L` by sampling items from their domains. The domain is dependent on `overlap_probability_function` as explained in Section 3.2.
4. Once S and L contain all elements from A and B respectively, add ties to the rankings as demonstrated in Section 3.3.
5. If required, truncate S and L to the desired length.

4 Results

To evaluate the effects of tuning the overlap probability function, we can plot the agreement A_d as depth d increases to the domain size n.

For example, we look at the impact of using the two different exponential decay functions, f from (9), to influence overlap probability. We show this using two different values of θ, thus varying the degree of 'top-weighted agreements'. The simulation algorithm is run 100 times and the average agreements at each depth is recorded and presented graphically in Figure 2.

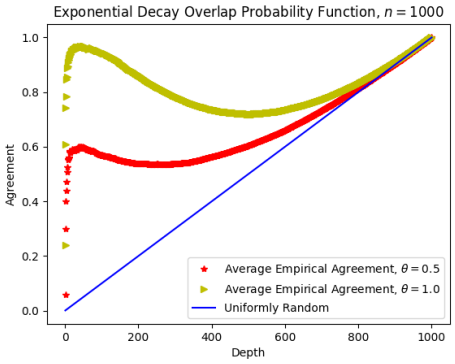

Figure 2: Plot of average ranking agreements as depth varies using two exponential decay functions with varying θ over 100 simulations.

Furthermore, we can also evaluate RBO_{EXT} on rankings generated from varying θ. From the formulaic definition of RBO, we expect that rankings where overlap is weighted towards the start of the rankings should be 'more similar' than rankings with overlaps spread throughout the ranking pair. To confirm this, we run 100 simulations of the simulation algorithm for each θ in `[0.0, 0.2, 0.4, ..., 1.0]` and plot the mean RBO_{EXT} for each θ. This scatter plot is shown in Figure 3.

In Section 3.2, we developed an analytical method to predict agreement behaviour given a function f. To verify its accuracy in predicting the actually observed empirical agreement, we plot the predicted agreement along with the average

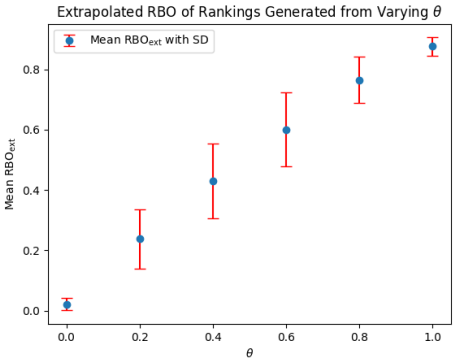

Figure 3: Scatter plot with a 1 standard deviation error bar showing the average RBO_{EXT} with $p = 0.95$ as θ increases from 0.0 to 1.0. The mean is calculated from 100 simulations of the algorithm generating rankings from a fully conjoint domain of length 1000.

of the empirical agreement when running the rank generation algorithm (without ties) 100 times in Figure 4.

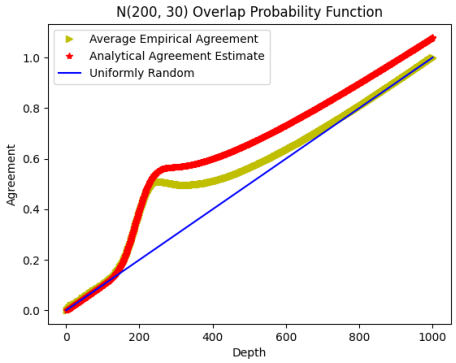

Figure 4: Plot of ranking agreement as depth increases. We can clearly see that the analytical estimate provides a close prediction to the empirically observed agreement when f is the Gaussian probability function centered at 200. For reference, the plot representing expected agreement between completely random rankings is also shown.

The algorithm also allows the user to tune the distribution of ties across the rankings, given a probability vector. To verify this behaviour, we can plot the number of tied items in each 'section' of the ranking when varying the tie probability function - Section 0 contains items between and including depths 1 and 10, Section 1 contains items between depths 11 and 20 etc.

We can observe the effect of using different distributions by counting the number of tied items in each section. Firstly, we simulate a ranking (we use the generated S, the ranking L is ignored), that has a uniform tie probability vector. We simulate this 100 times and compute the average number of tied items in each section. For reference, we run the simulation with `frac_ties_x = 0.6` and `n_groups_x = 40`. The result is shown in the plot in Figure 5.

One may also choose to simulate a ranking where tie probabilities are normally distributed around some depth d, with some standard deviation σ dictating the extent to which tie groups are spread out. Again, we run 100 simulations and compute the average number of tied items per section. In this case, we use a vector where probabilities are normally distributed around depth $d = 1500$ with standard deviation $\sigma = 500$. The plot is shown in Figure 6.

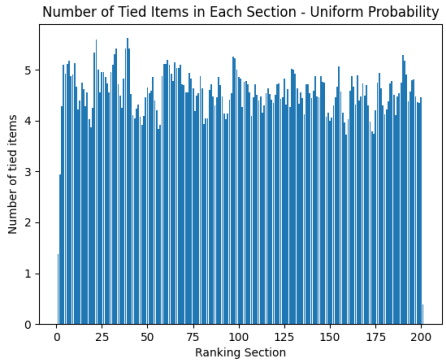

Figure 5: Plot of the average number of tied items in each 10-item section of the ranking when ties are distributed uniformly over 100 simulations.

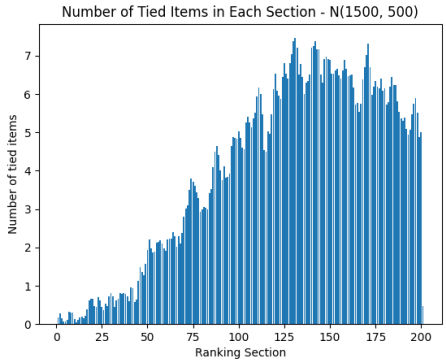

Figure 6: Plot of the average number of tied items in each 10-item section of the ranking when ties are distributed normally over 100 simulations.

We conclude this section by noting the qualitative advantages of the synthetic generation algorithm introduced in this paper over the method developed by Corsi and Urbano in [1].

Firstly, the new method offers the user a much greater degree of flexibility in shaping the distribution of ties within the ranking. This may be useful when considering RBO extrapolation for a ranking containing ties. In all three variants introduced in [5] and [1], RBO_{EXT} was calculated under the assumption that there are no ties in the unseen part. For a researcher looking to observe the implications if this assumption is relaxed, it may be useful to simulate rankings where ties occur after a certain depth l. As currently ties are created in an ad-hoc manner, a user wishing to create a dataset containing ties at specific positions may unsuccessfully run the algorithm several times before obtaining a satisfactory pair of

rankings.

Secondly, we see that this method of generating synthetic rankings natively allows for generating disjoint rankings given a desired `jaccard_similarity`. This is in contrast to Corsi and Urbano's method which inherently generates rankings from a fully conjoint domain where a user desiring some disjointness must truncate the ranking to a length orders of magnitude lower than the size of the domain.

A key aspect missing from Corsi and Urbano's algorithm was the user's lack of ability to influence the overlap structure of the generated ranking pairs beyond the `tau` parameter, which can only determine the correlation of the two rankings overall but cannot offer variation in the location of overlap within the rankings. Using the proposed method, by supplying an `overlap_probability_function`, the user can easily influence the similarity of the two rankings as depth varies.

5 Responsible Research

To ensure that results are reproducible, we note the inputs that are used to generate each plot in Section 4. Furthermore, all source code used and created in the process of research is openly available and accessible. This includes the Jupyter Notebook script `03-results.ipynb` used to generate the results presented in Section 4.

This paper also contributes to improving the nature of future research. Researchers looking to develop their own rank similarity measures can use the method presented in this paper to generate synthetic data, rather than using real datasets. Using synthetically generated data as opposed to real data alleviates the risk of encountering any ethical and legal issues such as privacy regulations and data protection laws.

In the interests of open and accessible science, all code was managed using a version control system. Initially, the TU Delft Computer Science department's hosted GitLab instance was used until the algorithm was complete. After that, the entire source code has been moved to a public GitHub repository. Furthermore, the repository is licensed using the MIT license, effectively allowing anyone to freely modify and distribute the software.

6 Discussion

Firstly, we analyse the plot presented in Figure 2. We expect that an exponential decay probability function with large θ should have a high level of agreement at earlier depths while a function with a lower, non-zero θ will have a lower level of agreement, but still greater than the expected level of agreement a random permutation of the domain's items. We see that for $\theta = 1$, the average agreement approaches 1.0 at a very early depth, before decreasing gently to around 0.7 and then again increasing back to 1.0. A similar behaviour is observed when $\theta = 0.5$, however the agreement peaks at just under 0.6 at the start of the rankings. This plot reflects our initial intuition.

In Section 2, we discussed that Rank-Biased Overlap evaluates ranking pairs with top-weighted overlap more favourably than a ranking pair where overlap is randomly spread throughout. Thus, we expect the ranking pair in yellow ($\theta = 1$) to have a higher RBO score than the ranking pair in red ($\theta = 0.5$). This is quantified more formally by plotting RBO_{EXT} for ranking pairs generated using the exponential decay function with varying θ. We also perform a linear regression statistical test to test the null hypothesis that there is no correlation between θ and RBO_{EXT} using the `scipy.stats.linregress` package [7]. We get a p-value of $3.9459 \cdot 10^{-5}$, meaning we can reject this null hypothesis with a reasonably high level of confidence.

Figure 4 displayed the average agreement observed between two rankings generated from a simulation where f was the scaled Gaussian probability density function (in red) along with its analytical estimate (in yellow). We see that while the analytical estimate reveals the innate characteristics observed empirically (deviation from agreement between the 'control' blue line) at exactly depth $d = 200$, it still overestimated the agreement significantly. This stems from the assumption that there will always be at least one element in each derived set for each of the four cases identified to increase overlap.

Figures 5 and 6 demonstrate the effects of varying the tie probability distribution. When the tie probability distribution is uniform, we see that the bar plot showing the number of tied items across each ranking section is uniform. On the other hand, Figure 6 shows how the number of tied items per ranking section varies when a normal distribution is used. This plot shows the familiar normal distribution shape, peaking at around section 150 as expected.

7 Conclusions and Future Work

In this paper, we introduced a novel method to generate a pair of rankings parametrised by three key properties: domain conjointness, ranking overlap structure and the distribution of ties. We also discussed an analytical method for the researcher to forecast the agreement of the synthetic rankings S and L given their overlap probability function f.

Firstly, we critically analysed the existing ranking generation algorithm and identified certain limitations that could be improved upon. We then thoroughly examined the method's ability to parametrise each of the key three ranking properties.

Using Jaccard simililarity as a domain conjointness parameter was a natural choice as its basis in set similarity offers a logical connection to RBO's view of ranking similarity through the prism of set overlap rather than correlation between ordinal datasets. The user influences the nature of ranking overlap through the `overlap_probability_function`: a function f which takes as input the depth d and outputs a probability $f(d)$ of increasing overlap. Ties are introduced once the base rankings have been created. The location of tie groups are influenced by the `tie_probabilities` vector.

Subsequently, we demonstrated the efficacy of the ranking generation method quantitatively, through several plots demonstrating that the algorithm performs as expected when subjected to different inputs. We also highlighted some qualitative advantages of this method when compared to the algo-

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

rithm presented in [1]. Additionally, we noted the measures taken to ensure that the research process was conducted in a responsible manner, specifically discussing reproducibility and the ethical implications of the work presented in the paper.

In conclusion, this paper presents an algorithm allowing for more fine-grained control over synthetically generated ranking as compared to the method presented by Corsi and Urbano in [1]. Researchers looking to develop new rank similarity measures, or extend existing techniques, can use this algorithm to generate ranking pairs exhibiting a variety of characteristics, ranging from the degree of conjointness of the domains to the presence and distribution of ties.

We finish this paper by recommending future points of improvements:

- currently the probability of selecting each case is equal, this should be adapted to avoid a situation where an empty set is being sampled.
- the analytical estimate can be improved to incorporate the probability of a set being empty or not.
- the analytical formula was only developed for fully conjoint domains. A more generalised formula should be devised, taking into account the three parametrisable properties discussed in the paper.
- the current method to introduce ties into the ranking uses a Poisson distribution to determine the size of a tie group. It is possible that this is too restrictive for the user, who may wish to also have the ability to influence the size of a tie group; in this case, we recommend a future alteration to the algorithm to allow the user to input a tie group size probability function.

References

[1] Matteo Corsi and Julián Urbano. The Treatment of Ties in Rank-Biased Overlap. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, Washington, D. C., 2024.

[2] Paul Jaccard. Nouvelles recherches sur la distribution florale. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 44:223–70, 01 1908.

[3] M.G. Kendall. *Rank Correlation Methods*. C. Griffin, 1948.

[4] Prem Sagar Sharma, Divakar Yadav, and Pankaj Garg. A systematic review on page ranking algorithms. *International Journal of Information Technology (Singapore)*, 12:329–337, 6 2020.

[5] W Webber, A Moffat, and J Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst*, 28, 2010.

[6] Emine Yilmaz, Javed A. Aslam, and Stephen Robertson. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, page 587–594, New York, NY, USA, 2008. Association for Computing Machinery.