Revisiting Recursive Inversion Models for Permutations

Luca Weihs

Department of Statistics, University of Washington Seattle, WA, 98195, USA

Abstract

Developing good generative models of rankings has become an increasingly important topic given recent interest in learning human preferences from data, especially in the context of web-based search. The well known Mallows Model and Generalized Mallows Model provide elegant exponential family models on the space of rankings while remaining computationally tractable. Extending this work, Meek and Meilă (2014) developed Recursive Inversion models (RIMs) which generalize the prior models, improving flexibility while maintaining computational efficiency. We give a comprehensive review of RIMs and the associated methods for inference.

1 Introduction

Datasets containing rankings of items arise naturally from surveys of human preferences and more recently as computer based outputs in the context of web search. Growing interest among scientists and businesses to understand these types of datasets has highlighted the importance of producing statistical models of the generative processes underlying the production of ranked data. As the space of rankings is super-exponential, there are $n!$ possible rankings for n items, it is especially important to balance the expressiveness of the model with the tractability of inference; moreover, since we often desire insight into the process producing the rankings (e.g. are preferences between political candidates grouped by party affiliation?), the parameters of the model should be interpretable and informative.

A popular tool for building models on rankings is the inversion distance d_{inv}, a measure of the number of pairwise differences between two rankings. In particular, the well known Mallows model (Mallows, 1957) defines the probability of a ranking π as being proportional to $e^{-\theta d_{\text{inv}}(\pi,\pi_0)}$ where $\theta \in \mathbb{R}$ and $\pi_0 =$ (a reference ranking) are parameters. While the Mallows model has many desirable properties stemming from its simplicity, it suffers from limited

flexibility. Improving upon this deficit, Fligner and Verducci (1986) defined Generalized Mallows Models (GMMs) which contain the Mallows model as a subset. GMMs generate rankings by iteratively inserting elements into a list with the probability of inserting an item e into a particular location being controlled by an element-specific inversion distance and a real-valued parameter θ_e. GMMs retain many of the nice computational properties of the Mallows model while substantially increasing its applicability.

Meek and Meilă (2014) build further upon this prior work by introducing Recursive Inversion Models (RIMs), an exponential family model on rankings that generalizes the insertion procedure of GMMs. RIMs begin a with rooted binary tree structure having n leaves, corresponding to the ranked elements, and $n-1$ internal nodes, each with a corresponding real-valued parameter. Rankings are then generated by recursively moving from the leaves of the tree to the root performing what Meek and Meilă (2014) call a stochastic merge sort at each internal node that interleaves the partial rankings already produced at the node's left and right subtrees. One benefit of this approach is that it can give insight as to how different groups of elements are preferred over others, something that is not possible with GMMs.

If the underlying tree structure, or even just the ordering of elements associated with the trees leaves is known, then inference in RIMs is straightforward and corresponds to solving univariate convex minimization problems and running a procedure taking $O(n^4)$ time. In the case when nothing is known about the underlying tree structure then, due to the large number of rooted binary trees with labeled leaves, an exhaustive search to find the true maximum likelihood (ML) tree structure is computationally infeasible. Instead Meek and Meilă (2014) perform a simulated annealing search over the space of binary trees to find a local optimum of the likelihood. While the results of this search are not guaranteed to be globally optimal they tend to be quite good in practice as will be explored in Section 4.

This report will reproduce the theoretical and experimental results of Meek and Meilă (2014) with the goal of adding another perspective to the work and filling in details that were originally left inexplicit. In Section 2 we will give a complete mathematical description of RIMs and demonstrate how the Mallows model and GMMs fit within the RIM framework. Section 3 will consider some the theoretical aspects of inference in RIMs, with attention

placed on understanding when RIMs are identifiable, and also presents efficient algorithms for ML estimation. Application to real-world and simulated data sets will be considered in Section 4 and we conclude with a discussion of RIMs and possible future work in Section 5.

2 Recursive Inversion Models

Let $A = \{a_1, ..., a_n\}$ be a set of n elements and S_n be the set of all permutations of the elements in A, that is $S_n := \{\pi \in A^n : \pi_i \neq \pi_j,\ \forall\ 1 \leq i < j \leq n\}$. Every $\pi \in S_n$ induces a ranking on the elements of A and, in particular, for all $1 \leq i, j \leq n$ we will say that $a_i <_\pi a_j$ if a_i precedes a_j in π. For $\pi, \pi_0 \in S_n$ we then say that $a_i <_\pi a_j$ are *inverted* in π with respect to π_0 if $a_j <_{\pi_0} a_i$; note that this definition is not symmetric. We may readily encode all of the inversions between π and π_0 in an $n \times n$ matrix $D(\pi, \pi_0)$ called the *discrepancy matrix* where for all $1 \leq i, j \leq n$ we define

$$D_{ij}(\pi, \pi_0) := \begin{cases} 1 & \text{if } a_i <_\pi a_j \text{ and } a_j <_{\pi_0} a_i \\ 0 & \text{if otherwise} \end{cases} . \tag{1}$$

In the special case that π_0 is the identity permutation $D(\pi, \pi_0)$ is lower triangular.

Both the Mallows model and GMMs are defined using measures of distance built from D. The Mallows model relies on the *inversion distance* between π and π_0, defined as as $d_{\text{inv}}(\pi, \pi_0) := \sum_{1 \leq i,j \leq n} D_{ij}(\pi, \pi_0)$, the total number of pairs in π inverted with respect to π_0. In particular, given a fixed reference permutation $\pi_0 \in S_n$ and a real-valued parameter $\theta \in \mathbb{R}$ the Mallows model defines the probability of a ranking $\pi \in S_n$ as $P(\pi \mid \pi_0, \theta) = \frac{1}{Z_M(\theta)} \exp(-\theta d_{\text{inv}}(\pi, \pi_0))$. Here the normalization function $Z_M(\theta)$ does not depend on the reference permutation making the model particularly simple to compute. From this formulation however, we see that the Mallows model is very coarse, there is only a single θ parameter and the inversion distance between $\pi \in S_n$ and π_0 will be equal for many distinct π. This problem is partially addressed by the GMMs of Fligner and Verducci (1986).

GMMs introduce a more granular measure of distance based on the discrepancy matrix D; for every $1 \leq j \leq n$ and $\pi, \pi_0 \in S_n$, define $w_j(\pi, \pi_0) := \sum_{a_j <_{\pi_0} a_i} D_{ij}(\pi, \pi_0)$. Then for a collection of real valued parameters $\theta_1, ..., \theta_n \in \mathbb{R}$ and a fixed reference permutation π_0,

GMMs define $P(\pi \mid \pi_0, \theta) = \frac{1}{Z_{\text{GMM}}(\theta,\pi_0)} \exp(-\sum_{i=1}^{n} \theta_i w_i(\pi, \pi_0))$. GMMs can be understood generatively as an iterative insertion procedure where elements $a_k \in A$ are inserted into a ranking at a position with probability controlled by θ_k and the number of inversions the insertion would produce relative to the reference ranking. As with the Mallows model, the normalization constant $Z_{\text{GMM}}(\theta, \pi_0)$ can be computed efficiently.

The RIMs of Meek and Meilă (2014) continue this trend of increasing generalization by modifying the insertion procedure of GMMs. RIMs begin with a rooted binary tree τ with n leaves each associated with a unique element of A. We denote the $n-1$ internal nodes of τ as $\mathcal{I} = \{i_1, ..., i_{n-1}\}$ and define i_j to equal the triple $(\theta_j, \tau_L(i_j), \tau_R(i_j))$ where $\tau_L(i_j), \tau_R(i_j)$ are, respectively, the left and right subtrees of i_j, and $\theta_j \in \mathbb{R}$ is a real-valued parameter. Traversing the tree τ in preorder we find a reference ranking $\pi_\tau \in S_n$ corresponding to the order in which the leaves of τ are visited; visually this corresponds to reading off the leaves of τ from left to right. For simplicity we will denote a RIM with tree structure τ, parameters $\theta = (\theta_1, ..., \theta_{n-1})$, and reference permutation π_τ, compactly as $\tau(\theta)$. The generative process underlying RIMs is best understood recursively moving from the leaves of the tree to the root producing *partial-rankings*, rankings of a subset of the elements in A, along the way:

(i) As a base case suppose we are at a leaf node. Then there is no randomness and the partial-ranking produced is simply the singleton element corresponding to the leaf node.

(ii) Now suppose we are at an internal node i_j where we have randomly generated sub-rankings $\pi_L = (c_1, ..., c_l)$ and $\pi_R = (d_1, ..., d_r)$ at the left and right subtrees of i_j. Letting $L_j, R_j \subset A$ be the elements corresponding to the leaves in the left and right subtrees of τ respectively, we have that π_L, π_R are rankings on L_j and R_j. We then randomly generate a new partial-ranking π_{new} on $L_j \cup R_j$ which merges π_L and π_R while maintaining the relative ordering of elements within π_L and π_R, with the probability of π_{new} equalling

$$P(\pi_{\text{new}} \mid \tau(\theta)) \propto \exp(-\theta_j v_j(\pi_{\text{new}}, \pi_\tau))$$

where $v_j(\pi_{\text{new}}, \pi_\tau) := \sum_{a_k \in L_j} \sum_{a_\ell \in R_j} D_{\ell k}(\pi_{\text{new}}, \pi_\tau) =$ the number of inversions created with respect to π_τ by the merge.

(iii) Once we have performed (ii) at the root node the process has generated a full ranking $\pi \in S_n$.

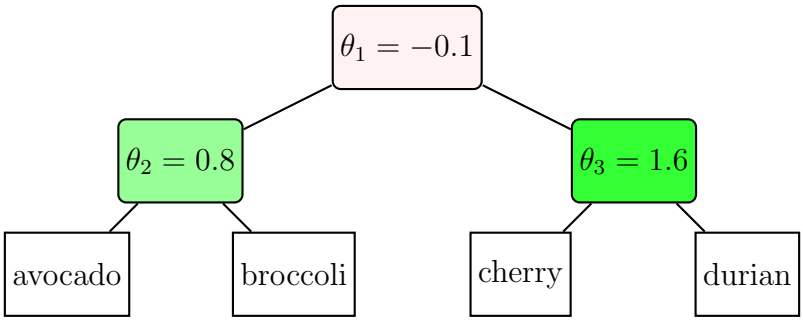
After performing this generative process it can be shown that the probability of a permutation π has the simple form

$$P(\pi \mid \tau(\theta)) \propto \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau)). \tag{2}$$

While the above generative process is useful for intuition it does not immediately lead to an algorithm for sampling from a given RIM as it is unclear how to efficiently perform the sampling in (ii). Indeed, actually sampling from a RIM is non-trivial and is not discussed in detail by the original paper of Meek and Meilă (2014). For those interested, we give a full overview of sampling in Appendix A.

Remark 2.1. Note that it is possible to represent a GMM as a RIM where the tree has a chain like structure, each internal node having a leaf as its left subtree. The advantages of RIMs over GMMs are emphasized by the following example.

Example 2.2. Consider the following RIM on the four elements (a)vocado, (b)roccoli, (c)herry, (d)urian. This RIM has reference permutation $\pi_\tau = (a, b, c, d)$ and *modal permutation*, the permutation with highest probability, $\pi_m = (c, d, a, b)$. We have that $\pi_\tau \neq \pi_m$ due to the negative value of θ_1 which causes c, d to be preferred over a, b. For the test permutation $\pi = (d, a, b, c)$ we have that $v_1(\pi, \pi_\tau) = 2, v_2(\pi, \pi_\tau) = 0$, and $v_3(\pi, \pi_\tau) = 1$. Note that the model represents strong preferences of a over b and c over d but only captures week preferences between all other pairs. An interpretation in this case may be that avocados are preferred over broccoli and cherries preferred over durians but it is difficult to compare



fruits to vegetables so there is only a weak preference for fruits over vegetables. These types of preferences cannot be captures by GMMs since in GMMs a strong preference of a over b and c over d will induce a strong preference within (a, d) or (c, b).

While RIMs may seem complex, the following proposition shows that the normalization (partition) function associated with Equation (2) is simple to compute despite it being, naively, a sum over $n!$ terms.

Proposition 2.3. *Let* $Z(\tau(\theta)) := \sum_{\pi \in S_n} \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau))$ *be the partition function associated with Equation* (2)*. For every* $i_j \in \mathcal{I}$ *let* L_j, R_j *be the set of elements corresponding to the leaves of the right and left subtrees of* i_j*. Then,*

$$Z(\tau(\theta)) = \prod_{i_j \in \mathcal{I}} Z_{|L_j|,|R_j|}(\theta_j), \text{ where} \tag{3}$$

$$Z_{n,m}(\widetilde{\theta}) := G(n, m, e^{-\widetilde{\theta}}) = \frac{(e^{-\widetilde{\theta}})_{n+m}}{(e^{-\widetilde{\theta}})_n (e^{-\widetilde{\theta}})_m}, \tag{4}$$

and $(p)_k := \prod_{i=1}^{k}(1 - p^i)$*. The function* $G(n, m, p)$ *is called the Gaussian Polynomial (Andrews, 1976). Note that* $G(n, m, p)$ *is undefined when* $p = 1$ *and, in this case, we define it to equal its limit as* $p \to 1$ *so that* $Z(n, m, 0) = G(n, m, 1) := \binom{n+m}{n}$.

Proofs for the above and several of the following propositions are omitted as they are somewhat lengthy and would detract from the purpose of this paper.

3 Efficient Inference

Given a dataset $\mathcal{D} = \{\pi_1, ..., \pi_m\}$ of rankings there are three primary inferential tasks which we would like to be able to accomplish with RIMs. Ordered by increasing difficulty these tasks are:

(i) Estimate θ given a known tree structure τ and reference permutation π_τ.

(ii) Estimate the tree τ given a known reference permutation π_τ.

(iii) Estimate the tree τ and reference permutation π_τ simultaneously.

Within the framework of ML estimation framework, Meek and Meilă (2014) give exact methods by which to accomplish (i), (ii), and an approximate, randomized search based method, for accomplishing (iii).

Finding the ML estimates $\widehat{\theta}_1, ..., \widehat{\theta}_{n-1}$ given a tree τ is very straightforward. From Equation 2 we may write the averaged negative log-likelihood of $\mathcal{D}$ as

$$-\frac{1}{|\mathcal{D}|} \log P(\mathcal{D} \mid \tau, \theta) = \sum_{i_j \in \mathcal{I}} \left(\theta_j \overline{V}_j + \log Z_{|L_j|,|R_j|}(\theta_j) \right) \tag{5}$$

where $\overline{V}_j = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{m} v_j(\pi_i, \pi_\tau)$. From this expression of the negated log-likelihood it is readily apparent that the RIM, with τ, π_τ fixed, is a exponential family model with sufficient statistics $\{\overline{V}_j\}$. As such, the log-partition function is convex in θ. Moreover, since Equation (5) decomposes into a sum of $score(j, \theta_j) := \theta_j \overline{V}_j + \log Z_{|L_j|,|R_j|}(\theta_j)$ over $1 \leq j \leq n-1$, it is easy to see that finding the MLEs $\widehat{\theta}_1, ..., \widehat{\theta}_{n-1}$ amounts to solving $n-1$ univariate convex minimization problems. Thus accomplishing (i) is simple by standard gradient descent methods. We should note that there are cases in which $score(j, \theta_j)$ does not have a global minimum, for instance when $|L_j| = |R_j| = 1$ and $\overline{V}_j = 0$ we have that $score(j, \theta_j)$ is positive everywhere but approaches 0 as $\theta_j \to \infty$. To resolve this problem we will always assume θ takes value in a compact hypercube of the form $[-M, M]^{n-1}$ where M is a large positive number; similarly we could have placed a prior upon θ and performed maximum a posteriori (MAP) estimation or, instead, added a small amount of noise to the $\overline{V}_j$ values before performing the ML optimization.

For the remainder of this report we will let $\overline{D} = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{m} D(\pi_i, \pi_{\text{id}})$ be the $n \times n$ matrix representing the average of all discrepancy matrices between rankings from the data and the identity permutation $\pi_{\text{id}} = (a_1, ..., a_n)$. One may readily compute the sufficient statistics $\overline{V}_j$ from $\overline{D}$ by noting that

$$\frac{1}{|\mathcal{D}|} \sum_{i=1}^{m} D_{ij}(\pi_m, \pi_\tau) = (1 - \overline{D}_{ij}) \cdot D_{ij}(\pi_\tau, \pi_{id}) + \overline{D}_{ij} \cdot (1 - D_{ij}(\pi_\tau, \pi_{id})).$$

Before describing how to accomplish tasks (ii) and (iii) above we will first explore some identifiability issues with RIMs which will demonstrate the lack of uniqueness in the MLEs

Algorithm 1

procedure CANONICALPERMUTATION($\tau(\theta)$)
  for $j \leftarrow 1, 2, ..., n-1$ **do**
    if $\theta_j < 0$ **then**
      $\theta_j \leftarrow -\theta_j$
      Flip the subtrees $\tau_L(i_j)$ and $\tau_R(i_j)$
  return $\tau(\theta)$

of τ and π_τ and motivate the definition of a canonical form for τ.

3.1 Identifiability

Given a RIM $\tau(\theta)$ there are two fundamental identifiability problems. The first occurs when flipping the sign of one of the parameters θ_j while simultaneously flipping the left and right subtrees $\tau_L(i_j)$ and $\tau_R(i_j)$. This unidentifiability is summarized in the following proposition.

Proposition 3.1. *Let $\tau(\theta)$ be a RIM. Let $i_j = (\theta_j, \tau_L(i_j), \tau_R(i_j))$ be an internal node of $\tau(\theta)$ and let $\widetilde{\tau}(\widetilde{\theta})$ be $\tau(\theta)$ but with i_j replaced by $\widetilde{i}_j = (-\theta_j, \tau_R(i_j), \tau_L(i_j))$. Then for all $\pi \in S_n$ we have $P(\pi \mid \tau(\theta)) = P(\pi \mid \widetilde{\tau}(\widetilde{\theta}))$.*

From this Proposition it follows immediately that for any RIM $\tau(\theta)$ there exist at least $2^{n-1} - 1$ other RIMs which generate the same probability distribution. As we cannot hope to distinguish these models with data, Meek and Meilă (2014) choose to group all such models into an equivalence class and then definite a canonical representative of that class which we often can recover. In particular, a RIM $\tau(\theta)$ with $\theta_1, ..., \theta_{n-1} \neq 0$ is said to be in *canonical form* if $\theta_1, ..., \theta_{n-1} > 0$. The restriction to RIMs with $\theta_1, ..., \theta_{n-1} \neq 0$ is intuitively reasonable as if some $\theta_j = 0$ then there is no preference between the two sets of elements at the leaves of each subtree $\tau_L(i_j), \tau_R(i_j)$ and so forcing a particular ordering upon them by some other criterion is unnatural. There is a trivial algorithm for placing a RIM τ into canonical form presented in Algorithm 1. When there is a unique canonical form inducing the same probability distribution as a RIM $\tau(\theta)$, we will say that $\tau(\theta)$ is *identifiable*.

The second unidentifiability problem occurs when certain values of θ_j are equal. For instance if there exists some $\widetilde{\theta} \in \mathbb{R}$ so that $\widetilde{\theta} = \theta_1 = \cdots = \theta_{n-1}$, then the probability of a

permutation π is

$$P(\pi \mid \tau(\theta)) \propto \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau)) = \exp(-\widetilde{\theta} \sum_{i_j \in \mathcal{I}} v_j(\pi, \pi_\tau)) = \exp(-\widetilde{\theta} d_{\text{inv}}(\pi, \pi_\tau)),$$

so that the RIM reduces to being a Mallows model. In this case it is clear that every tree structure having the same reference permutation induces the same probability distribution. This type of unidentifiably is characterized by the following definition. We will say that a RIM $\tau(\theta)$ is *locally identifiable* if for every internal node i_j of τ we have that $\theta_j \neq 0$ and $|\theta_k| \neq |\theta_j|$ for every child i_k of i_j. Surprisingly, local identifiability and identifiability correspond exactly.

Proposition 3.2. *A RIM $\tau(\theta)$ is identifiable if and only if it is locally identifiable.*

3.2 Estimating τ when π_τ is known

Suppose we have fixed a reference permutation π_τ so that we know the order of the order of the leaves of the RIM. Then Meek and Meilă (2014) show that, by using a dynamic programming strategy, it is possible to find the ML tree structure τ in $O(n^4)$ time. Here this runtime assumes that it is possible to find the MLE $\widehat{\theta}_j$ by solving the univariate convex minimization task described earlier in a constant number of iterations. Empirically we found this assumption to be reasonable as convergence tends to occur in < 100 iterations of a simple gradient descent algorithm. We summarize this result in the following proposition.

Proposition 3.3. *Given a set of elements A, a reference permutation π_τ of A, and a collection of observed data $\mathcal{D}$, it is possible to find the ML RIM tree structure $\widehat{\tau}$ in $O(n^4)$ time using Algorithm 2 (*StructByDP*).*

The proof of Proposition 3.3 strongly reminiscent of the proofs for other dynamic programming algorithms; for instance, the algorithm is almost identical to both the optimal matrix chain multiplication algorithm (Cormen, Leiserson, Rivest, and Stein, 2009) and Wagner-Fischer algorithm for computing the edit distance between two strings (Wagner and Fischer, 1974). As the proof is intuitive and instructive we include it in Appendix B.

Algorithm 2

procedure StructByDP($\pi_\tau, \mathcal{D}$)
  $\overline{D} \leftarrow$ the average discrepancy for the data $\mathcal{D}$
  for $m = 1, ..., n$ **do**
    $cost(m, m) \leftarrow 0$
  for $l \leftarrow 2, ..., n-1$ **do**
    for $j \leftarrow 1, ..., n-l+1$ **do**
      $m \leftarrow j + l - 1$
      $cost(j, m) \leftarrow \infty$
      for $k \leftarrow j, ..., m-1$ **do**
        $\overline{V} \leftarrow \sum_{j'=j}^{k} \sum_{m'=k}^{m} \overline{D}_{m'k'}$ ▷ See the proof of Proposition 3.3 in Appendix B for how to compute $\overline{V}$ efficiently
        $\widehat{\theta} \leftarrow \arg\min_\theta score(j, k, m, \theta)$
        $s \leftarrow score(j, m) + cost(j, k) + cos(k+1, m)$
        if $s < cost(j, m)$ **then**
          $cost(j, m) \leftarrow s$
          $back(j, m) \leftarrow k$
  Using *back* generate the ML tree $\widehat{\tau}(\widehat{\theta})$ where $\widehat{\theta}$ are are MLEs of θ
  return $\widehat{\tau}(\widehat{\theta})$

3.3 Estimating τ via local search

From Section 3.2 we are able to efficiently compute the ML tree structure $\widehat{\tau}$ when the reference permutation π_τ is known. When wishing to perform this estimation when π_τ is unknown a first approach would be to simply enumerate all of the possible reference permutations π_τ, perform the ML estimation for each of these permutations, and pick the tree that maximizes the likelihood. While this approach would be optimal it is computationally intractable as the number of permutation on n elements is $n!$. Instead of performing this enumeration exactly, Meek and Meilă (2014) use a stochastic search which they describe as simulated annealing, perhaps a slight misnomer as there is no actual annealing in their algorithm. This approach is similar to the well-performing greedy algorithms examined by Schalekamp and van Zuylen (2009) and Ali and Meilă (2012) which make small changes to a reference permutation and retain the changes which are locally optimal.

The first obstacle when performing the search is the initialization of the reference permutation. While it is certainly possible to initialize randomly or with a fixed, say lexicographic ordering, local search algorithms are often sensitive to their starting location and may find

local optimums which are poor with respect to the global optimum when initialized naively. Luckily there exist many efficient algorithms in the context of GMMs for finding well-fitting reference permutations and any of these algorithms may be used for initialization.

Following initialization we search by using StructByDP to produce a optimal RIM $\tau^*(\theta^*)$ for the current reference permutation and then sample from this RIM to produce a new candidate permutation π. From this candidate permutation we use StructByDP to create a new RIM, generate the canonical permutation π' of that RIM with CanonicalPermutation, and then use StructByDP once more on π' to generate another new RIM $\tau(\theta)$. We then test if $\tau(\theta)$ corresponds to a RIM with better log-likelihood than the previous RIM $\tau^*(\theta^*)$. If so we accept the new RIM and repeat the search procedure, if not then we accept the RIM with some probability controlled by a inverse temperature parameter β and the difference between the previous and current log-likelihoods. While the transformation to a canonical permutation and repeated application of StructByDP may seem redundant, Meek and Meilă (2014) note that doing so often substantially increases the log-likelihood in their experiments. Note that the above procedure is a Metropolis-Hastings algorithm where candidates are randomly generated and then accepted with some probability depending on the proportion of their likelihood to the likelihood of the current model (Hastings, 1970).

Choosing a new candidate permutation by sampling from the current RIM is intuitive as if for a current reference permutation π an ordering $a_i <_\pi a_j$ is unsupported by the data then it is likely to only be weakly ordered by the RIM produced by StructByDP. As such it is likely that the sampled permutation will not maintain that ordering. Similarly if $a_i <_\pi a_j$ is strongly supported by the data then the newly sampled permutation will likely maintain that ordering. The algorithm described above is called SASearch and is provided as pseudo-code in Algorithm 3.

4 Computational Experiments

In these experiments we will empirically assess the consistency properties, relative strength when compared to competing models, resilience to violations of assumptions, and computational efficiency of RIMs. We implemented RIMs and their associated algorithms in C++

Algorithm 3

procedure SASEARCH($\mathcal{D}$, inverse temperature parameter $\beta > 0$)
  $\overline{D} \leftarrow$ the average discrepancy for the data $\mathcal{D}$
  $\pi_0 \leftarrow$ reference permutation generated by an efficient GMM estimation algorithm
  $\tau_0 \leftarrow$ STRUCTBYDP($\pi_0, \overline{D}$)
  $\tau^{best} \leftarrow \tau_0$
  for $t = 1, 2, ..., t_{max}$ **do**
    while accept=FALSE **do**
      $\pi' \leftarrow$ sample from τ_{t-1}
      $\tau' \leftarrow$ STRUCTBYDP($\pi', \overline{D}$)
      $\tau' \leftarrow$ CANONICALPERMUTATION(τ')
      $\pi' \leftarrow$ reference permutation of τ'
      $\tau' \leftarrow$ STRUCTBYDP($\pi', \overline{D}$)
      $accept \leftarrow TRUE$
      $u \leftarrow$ sample from Uniform(0,1)
      if $\exp\left(-\beta\left(\log P(\overline{D} \mid \tau_{t-1}) - \log P(\overline{D} \mid \tau')\right)\right) < u$ **then**
        $accept \leftarrow FALSE$
    $\tau_t \leftarrow \tau'$
    if $P(\overline{D} \mid \tau_t) > P(\overline{D} \mid \tau^{best})$ **then**
      $\tau^{best} \leftarrow \tau_t$
  return τ^{best}

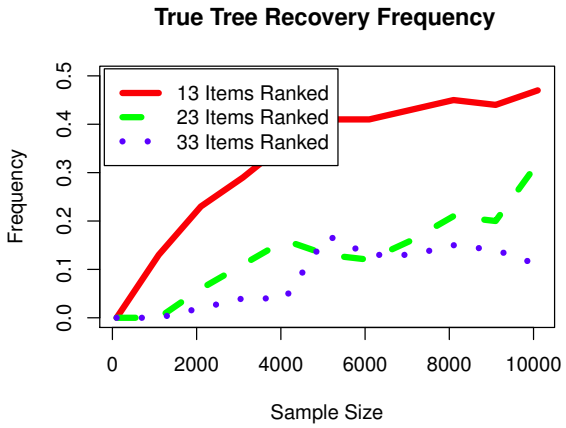
and interfaced this implementation with R using the Rcpp package (Stroustrup, 2000; R Core Team, 2014; Eddelbuettel and François, 2011). The experiments were then conducted through R with visualization of RIM tree structures aided by the igraph package (Csardi and Nepusz, 2006). Comparing our implementation to that from Meek and Meilă (2014) we find that a search which required them around 3 minutes takes us consistently less than 5 seconds, a > 30 fold improvement.

4.1 Empirical Consistency

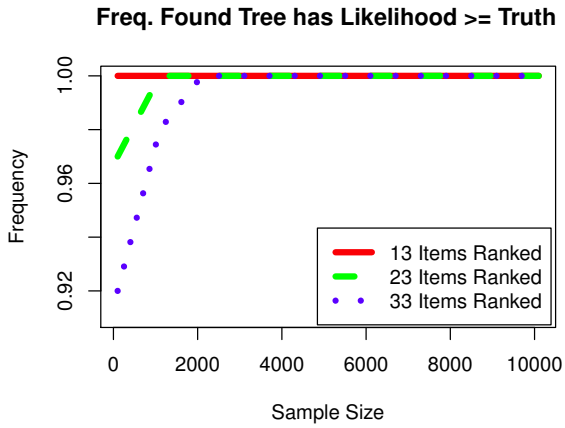
To understand the consistency properties of RIMs we, for each choice of $n \in \{13, 23, 33\}$ and $N \in \{100, 1100, ..., 10100\}$, performed the following procedure:

(1) Generate 100 random RIM tree structures $\tau_1, ..., \tau_{100}$ with n leaves and internal parameters $\exp(-\theta_i)$ randomly chosen from $(.4, .9)$ uniformly.

(2) For each RIM tree structure generate a sample of size N, call these samples $\mathcal{D}_1, ..., \mathcal{D}_{100}$.



(a) True tree discovery rate.



(b) Rate at which SASearch discovers a tree with likelihood on the data $\geq$ the true tree.

Figure 1: The proportion of times, out of 100, the true tree was correctly recovered by SASearch for a given sample size and number of items being ranked; also the proportion of times that the approximate ML RIM structure found by SASearch had a greater log-likelihood on the sample.

(3) For each $\mathcal{D}_i$ run the SASearch algorithm on $\mathcal{D}_i$ with inverse temperature parameter $\beta = 5$ and save the resulting approximate maximum likelihood tree as $\widehat{\tau}_i$.

(4) Record the proportion, out of 100, of the $\widehat{\tau}_i$ whose tree structure equals τ_i. Also record the proportion of times $\widehat{\tau}_i$ has a larger or equal likelihood on $\mathcal{D}_i$ than τ_i (where both trees use their maximum likelihood θ values).

The results of these computations are summarized in Figure 1. As Figure 1a shows, even when only 13 items are being ranked we only recover the full true tree approximately 47% of the time with 10,100 samples and this proportion is approximately 3/5 times as large for 23 items and approximately halved again for 33 items. Thus for relatively large sample sizes (indeed we could not find any publicly available datasets of complete ranking on items with more than 6000 samples) it does not seem hopeful to recover the true tree with high probability when the number of items being ranked is much beyond 13. Moreover Figure 1b shows that SASearch almost always discovers a tree structure whose likelihood on the sample is larger than that of the true tree structure; this suggests that the low true tree recovery rate is not a symptom of SASearch performing poorly but is instead due to space

of RIMs being sufficiently expressive that over-fitting a dataset is easy. Note also that this is not a case in which we may use information criteria (such as the AIC (Akaike, 1974) or BIC (Schwarz, 1978)) to aid in model selection since the number of θ parameters does not change for different tree structures. Thankfully it is often unnecessary to recover the true tree exactly and, in the examples below, we will see that certain important features of the data are consistently discovered by SASearch with relatively small sample sizes.

It is difficult to compare our results with those of Meek and Meilă (2014) as, for their simulation experiments, they do not provide any numerical results. They do, however, note that a large sample was generally required to recover the true tree and that failures in identification were generally due to there being tree structures with a larger likelihood than the truth, both of these observations agree with Figure 1.

4.2 Real-World Data Experiments

We run extensive experiments on two real-world datasets. The first dataset contains 5000 rankings of 10 different types of sushi collected by a questionnaire survey (Kamishima, 2003). The 10 types of sushi ranked are ebi (shrimp), anago (sea eel), maguro (tuna), ika (squid), uni (sea urchin), sake (salmon roe), tamago (egg), toro (fatty tuna), tekka-maki (tuna roll), and kappa-maki (cucumber roll). The second dataset contains 2490 fully ranked ballots cast during the 2002 Parliamentary elections of the Meath constituency in Ireland. Voting in Ireland is carried out using a system called proportional representation by means of a single transferable vote; voters rank in order of their preferences all, or a subset of, the candidates and the winners of the election are determined by a series elimination rounds, for more details on the dataset and the voting system see Gormley and Murphy (2007). In the 2002 election there were 14 candidates from 7 different parties running for 5 seats in the House of Parliament, the candidates and their associated parties are presented in Table 1.

Following the strategy of Huang and Guestrin (2012) we will attempt to understand the behavior of RIMs at small samples by repeatedly bootstrapping different sample sizes and comparing features of the inferred trees from the bootstrap samples to trees inferred when using all of the data. The RIM models inferred for both datasets when using all of the data, 1000 SASearch iterations, and an inverse temperature parameter of $\beta = 50$ are displayed in

Candidate	Party Affiliation
Brady, J	Fianna Fáil (FF)
Bruton, J.	Fine Gael (FG)
Colwell, J.	Independent (I)
Dempsey, N.	Fianna Fáil (FF)
English, D.	Fine Gael (FG)
Farrelly, J.	Fine Gael (FG)
Fritzgerald, B.	Independent (I)

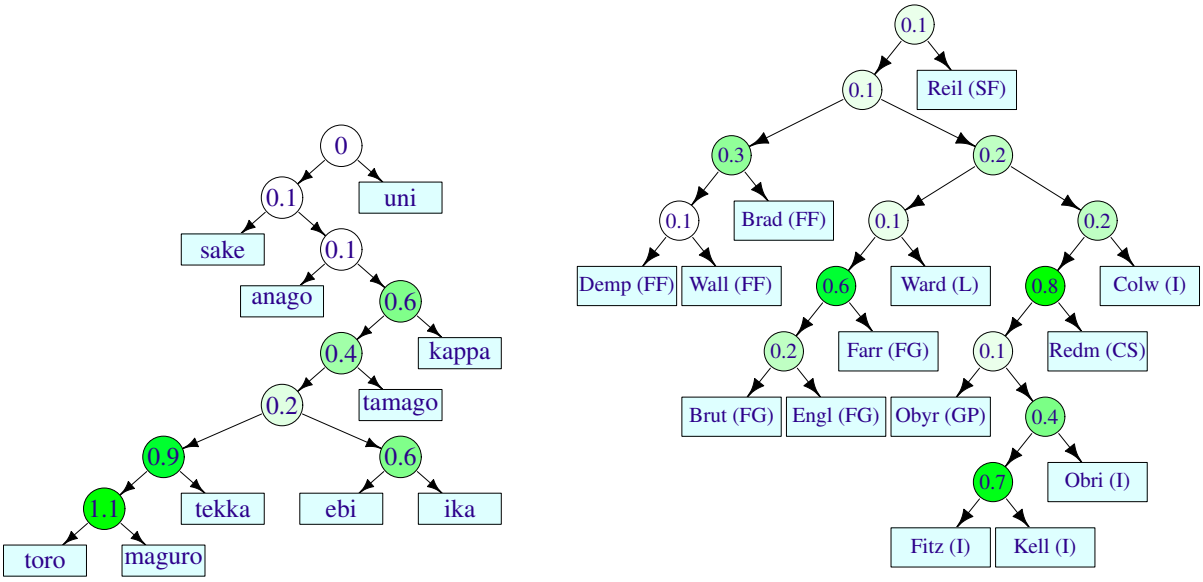
Candidate	Party Affiliation
Kelly, T.	Independent (I)
O'Brien, P.	Independent (I)
O'Byrne, F.	Green Party (GP)
Redmond, M.	Christian Solidarity (CS)
Reilly, J.	Sinn Féin (SF)
Wallace, M.	Fianna Fáil (FF)
Ward, P.	Labour (L)

Table 1: Candidates of the 2002 Meath constituency House of Parliament elections along with their party affiliations.

Figure 2. We will call these two trees the approximate ML trees for their respective datasets. Note that the tree recovered from the sushi data is exactly the same as the tree presented by Meek and Meilă (2014) in their Figure 2. From Figure 2 we see that there are several interpretable features of both learned models. For the sushi tree we see that at the top level uni (sea urchin) is broken off from the rest of the tree with a θ value near 0, this suggests that uni is not consistently comparable to the other types of sushi and may appear anywhere in a ranking. Meek and Meilă (2014) suggest this is intuitive as uni is often described as having a taste that is unique among other varieties of sushi. Moreover we see that toro (fatty tuna) is strongly preferred to maguro (tuna) and that ebi (shrimp) is moderately preferred over ika (squid).

For the RIM inferred from the Meath data we see, similarly to the sushi model, that at the top level a single leaf representing the candidate J. Reilly is broken off from the rest of the tree and has a small θ value associated with this break. Again this suggests that there is no consistent ranking of J. Reilly, an interpretation that agrees with the fact that J. Reilly is associated with the more radical Sinn Féin party. Futhermore, we see that the model groups most of the candidates with the same party affiliation in the same subtrees; indeed there are three subtrees of size 5 all of whose leaves are are associated with the Fianna Fáil, Fine Gael, and Independent parties respectively. This structure agrees with the common belief that preferences between members of different parties is strongly captured by just the preferences between parties.

We would now like to measure how rapidly the above features of the approximate ML trees



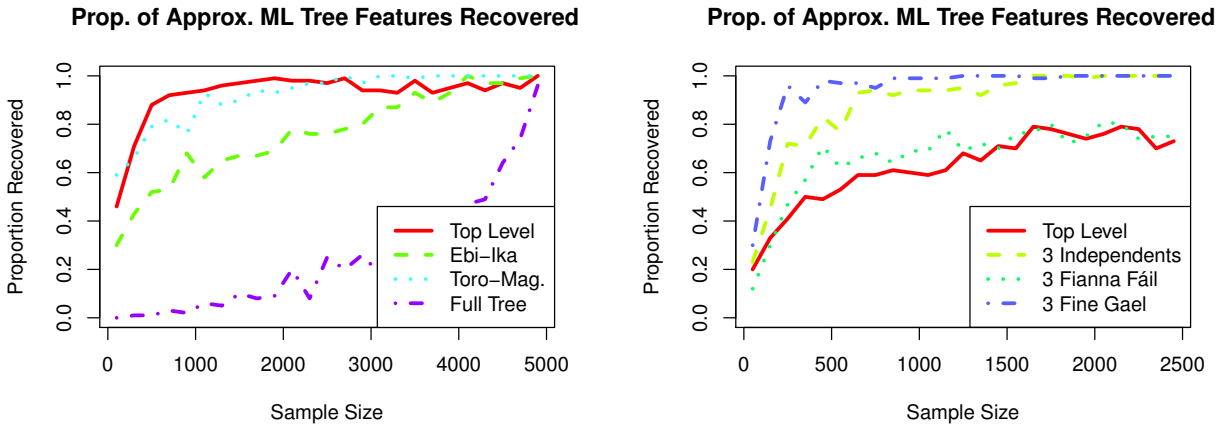
(a) Sushi RIM.

(b) Meath RIM.

Figure 2: The RIMs inferred by SASEARCH when using all of the available data from the two datasets, 1000 iterations, and an inverse temperature parameter of $\beta = 50$. The θ parameter values at internal nodes have been rounded to 1 decimal place.

are captured for increasingly large sample sizes. Taking inspiration form the experiments of Huang and Guestrin (2012) we, for various sample sizes N bootstrap 100 datasets of size N, run SASEARCH for 200 iterations with $\beta = 50$ on each bootstrap sample, and compute the proportion of times the above features appear in the samples. The reference permutations used to initialize the searches were generated by fitting GMM models to the bootstrapped data sets, for the sushi data the GMM model was fit using the BRANCH&BOUND algorithm of Mandhani and Meilă (2009) and for the Meath data we fit the GMM by an approximate minimization of Kendall distance using the PerMallows R package of Irurozki (2014). This difference in initialization strategy was simply due to the BRANCH&BOUND algorithm being quite slow when trained on the Meath data, requiring several minutes to fit a single GMM. The results of these computations along with precise feature descriptions are displayed in Figure 3.

From the figure we see that certain features are almost always captured even for very small sample sizes. The split of uni from the rest of the tree at the top level, for example, is



(a) Sushi features recovered.

(b) Meath features recovered.

Figure 3: The proportion of times (out of 100 bootstrap samples) certain approximate ML tree features were recovered at various sample sizes. For the sushi dataset these features are: Top Level (uni split from the tree at the top), Ebi-Ika (Ebi and Ika appear as children of the same node with Ebi preferred to Ika), Toro-Maguro (the same as Ebi-Ika but for Toro and Maguro), and full tree (the inferred tree equalled the tree from Figure 2a). For the Meath dataset these features are: Top Level (Reilly (SF) split from tree at the top), 3 Independents (3 independents are the only leaves of a subtree), 3 Fianna Fáil (same as the last feature but for Fianna Fáil), and 3 Fine Gael (same as the last feature but for Fine Gael).

recovered more than 90% of the time with only around 1000 samples. Note that we do not, for the Meath data, include the rate at which the tree from Figure 2b was discovered as it would be largely obscured by the legend; for reference, even at the largest sample size the tree from Figure 2b was only discovered in 67% of cases, this large variability is in line with what was observed by Meek and Meilă (2014) and is largely due to SASearch being often unable to find the global optimum in 200 iterations.

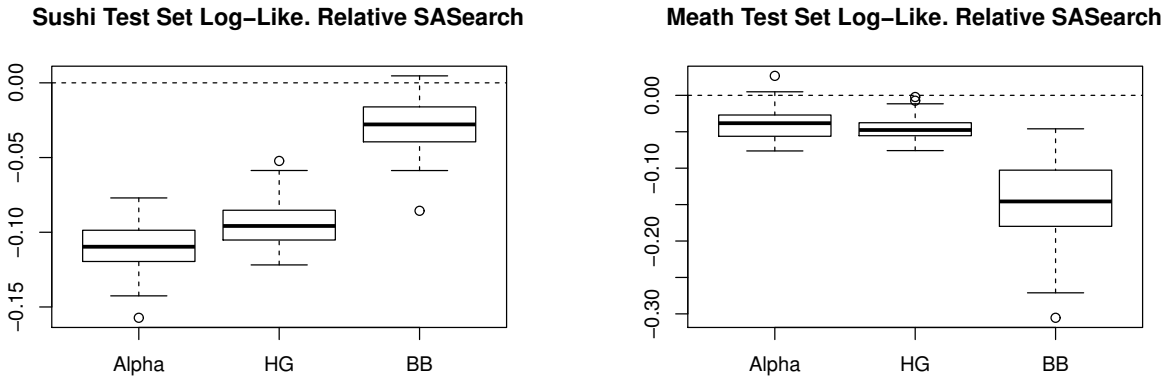
Finally it is of interest to compare the different methods for generating RIM structures given different assumptions on what is fixed in the model, as well as to compare against other models on rankings. We will use two different methods for generating RIM structures. The first, which we will call Alpha, simply takes the alphabetical ordering of rankings as the reference permutation and builds a RIM structure using StructByDP. The second will be to use SASearch with the same parameters as described above, initializing the reference permutation by using the output of the Branch&Bound algorithm. We will also compare against two other models for ranking. The first will be the GMM model found by

the Branch&Bound algorithm, we call this BB, and the second, which we will call HG, will be the riffled independence model of Huang and Guestrin (2012). Unfortunately it is not possible to compare directly with the riffled independence model as Huang and Guestrin (2012) have not made code publicly available, instead we will follow the practice of Meek and Meilă (2014) by fitting a RIM model to the hierarchical tree structure identified by Huang and Guestrin (2012). This HG comparison is flawed in a number of ways but will hopefully provide some measure of the goodness of the hierarchical structure identified by Huang and Guestrin (2012).

For each of the above methodologies we perform a randomized cross validation where we randomly sample 300 data points from the full dataset and treat the remaining data points as a test set, train each of the models on the 300 data points, and record the log-likelihood of each of the models on the test set. We perform this procedure 100 times for the Sushi data and 50 times for the Meath data and record the results in Figure 4. We see that SASearch results in a test set log-likelihood greater than the other models in almost all cases. For the Sushi data we see that the GMM produced by the Branch&Bound algorithm does quite well, outperforming both Alpha and HG. This suggests that the linear ordering induced by GMMs is not a poor model for the Sushi data. For the Meath data, however, we see that the BB model does considerably worse than the Alpha and HG. Meek and Meilă (2014) suggest that the poor performance of GMMs on the Meath dataset is due to GMMs inability to capture features such as the preferences between parties instead of between individuals that we discussed previously.

4.3 Violation of Assumptions

To better understand the performance of RIMs when assumptions are violated we ran a collection of simulations exploring the results of inference when samples were generated from a mixture of two RIMs and when outliers were introduced. In the bimodal case we found that RIMs performed as well as could be hoped, generally recovering features common to both RIMs or those features from the more likely mixture component. More surprisingly we found that, despite being an exponential family model where outliers generally can cause anomalous behavior, the features of RIM tree structure were often recovered even with large



(a) Sushi test set log-likelihoods.

(b) Meath test set log-likelihoods.

Figure 4: Test set log-likelihoods of Alpha, BB, and HG relative SASearch as box plots (positive values imply that the log-likelihood of the model was greater than that produced by SASearch). For the Sushi data we conducted 100 iterations of randomized cross validation with a training/test split of 300/4700. Similarly for the Meath data we conducted 50 iterations with a training/test split of 300/2190. For reference we have included a horizontal dashed line at 0.

numbers of outliers (up to 5% of the total sample size). We should note, however, that in the presence of outliers the estimated θ values often changed substantially.

5 Discussion

As an exponential family model generalizing GMMs and the Mallows model, the RIMs of Meek and Meilă (2014) are an important step in producing models on rankings which are interpretable, computationally tractable, and flexible. We have seen that RIMs reliably recover many important characteristics of ranking distributions with fairly small sample sizes, although recovery of the full RIM tree structure generally requires very large sample sizes. Our experimental results suggest that RIMs are most suited for producing a descriptive summary of a ranking distribution on small numbers of items, for instance in survey data of human preferences where we cannot expect humans to produce complete rankings of large numbers of items.

RIMs are closed related to, an indeed a subclass of, the hierarchical *Riffled Independence*

models (RI models) of Huang and Guestrin (2012). For a set of items $A = \{a_1, ..., a_n\}$ with a distribution h on their rankings, Huang and Guestrin (2012) define $B \subset A$ and $C = A \setminus B$ to be *riffle independent* under h if h can be generatively modeled as producing partial rankings σ_B, σ_C on B, C using distributions h_B, h_C and then interleaving σ_B, σ_C together using a riffle shuffle (a merge just as with RIMs) with some probability. The model then gains a hierarchical structure when assuming that h_B, h_C can be further decomposed into riffle independent sets recursively. Hierarchical RI models cover a very broad collection of distributions on rankings and indeed are easily seen to contain RIMs as a subset. This large flexibility, however, comes at the cost of RI models not being a collection of exponential family models; moreover learning hierarchical RI models often requires the creation of non-parametric estimates of distributions which can require memory exponential in the number of items. Finally, unlike RIMs, RI models are difficult to interpret and compare.

While we, in part, motivated the importance of understanding ranking distributions with the example of web search data it is, unfortunately, unclear how to apply RIMs to the domain of larger scale information retrieval. This inapplicability is largely due to what we consider the two major weakness of RIMs. Firstly, ranking data is often provided as collections of partial rankings while RIMs are a distribution over full rankings. For instance, in web search data we may only know that a user preferred a particular link (the link that was clicked) over all of the links that weren't clicked but were displayed. Thankfully, building off of the work of Huang, Kapoor, and Guestrin (2012) where it is shown that partial rankings can be used to learn hierarchical RI models, Meek and Meilă have shown in currently unpublished work that it is possible to learn RIMs from partial rankings. Secondly, the application of RIMs to web search is strongly hindered by $O(n^4)$ scaling of the StructByDP algorithm in the number of items n being ranked. There has, however, been work in extending GMMs to the case when there are infinitely many items to rank and data only includes rankings of the most preferred items (Meilă and Bao, 2010). This gives hope that such an efficient extension may be also found for RIMs.

References

H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, Dec 1974. doi: 10.1109/TAC.1974.1100705.

A. Ali and M. Meilă. Experiments with Kemeny ranking: what works when? *Math. Social Sci.*, 64(1):28–40, 2012. doi: 10.1016/j.mathsocsci.2011.08.008.

G. E. Andrews. *The theory of partitions*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1976. Encyclopedia of Mathematics and its Applications, Vol. 2.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `http://igraph.org`.

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `http://www.jstatsoft.org/v40/i08/`.

M. A. Fligner and J. S. Verducci. Distance based ranking models. *J. Roy. Statist. Soc. Ser. B*, 48(3):359–369, 1986.

I. C. Gormley and T. B. Murphy. A latent space model for rank data. In E. Airoldi, D. Blei, S. Fienberg, A. Goldenberg, E. Xing, and A. Zheng, editors, *Statistical Network Analysis: Models, Issues, and New Directions*, volume 4503 of *Lecture Notes in Computer Science*, pages 90–102. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-73133-7_7.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. doi: 10.1093/biomet/57.1.97. URL `http://biomet.oxfordjournals.org/content/57/1/97.abstract`.

J. Huang and C. Guestrin. Uncovering the riffled independence structure of ranked data. *Electron. J. Statist.*, 6:199–230, 2012. doi: 10.1214/12-EJS670.

J. Huang, A. Kapoor, and C. Guestrin. Riffled independence for efficient inference with partial rankings. *Journal of Artificial Intelligence Research*, pages 491–532, 2012.

E. Irurozki. *PerMallows: Permutations and Mallows distributions*, 2014. URL `http://CRAN.R-project.org/package=PerMallows`. R package version 1.7.

T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 583–588, New York, NY, USA, 2003. ACM. doi: 10.1145/956750.956823.

C. L. Mallows. Non-null ranking models. I. *Biometrika*, 44:114–130, 1957.

B. Mandhani and M. Meilă. Better search for learning exponential models of rankings. *Artificial Intelligence and Statistics AISTATS*, 12, 2009.

C. Meek and M. Meilă. Recursive inversion models for permutations. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 631–639. Curran Associates, Inc., 2014.

M. Meilă and L. Bao. An exponential model for infinite rankings. *The Journal of Machine Learning Research*, 11:3481–3518, 2010.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`.

F. Schalekamp and A. van Zuylen. Rank aggregation: together we're strong. In *ALENEX09—Workshop on Algorithm Engineering & Experiments*, pages 38–51. SIAM, Philadelphia, PA, 2009.

G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2):461–464, 03 1978. doi: 10.1214/aos/1176344136. URL `http://dx.doi.org/10.1214/aos/1176344136`.

B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2000.

R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, Jan. 1974. doi: 10.1145/321796.321811. URL http://doi.acm.org/10.1145/321796.321811.

Appendix A Generative Process for RIMS

As usual let $\tau(\theta)$ be a RIM with n leaves corresponding to the n items $A = \{a_1, ..., a_n\}$, internal nodes $\mathcal{I} = \{i_1, ..., i_{n-1}\}$, and parameters $\theta = (\theta_1, ..., \theta_{n-1}) \in \mathbb{R}^{n-1}$. Without loss of generality we will assume that the reference permutation of $\tau(\theta)$ is simply the identity, that is $\pi_\tau = (a_1, ..., a_n)$. Recall that the generative process for RIMs recursively moves from the leaves of the tree to the root producing partial rankings at each internal node by merging together the partial rankings produced at that nodes left and right children (leaf nodes always simply produce the singleton ranking corresponding to their item). What remains to be described is the random process by which partial rankings are merged at internal nodes that gives rise to the probability of a permutation π under $\tau(\theta)$ being

$$P(\pi \mid \tau(\theta)) \propto \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau)).$$

Suppose during the generative process we are at an internal node i_j and must merge together the partial rankings $\pi_B = (b_1, ..., b_m)$ and $\pi_C = (c_1, ..., c_q)$ produced at the left and right subtrees of i_j. That is we must generate a new partial ranking π_{new} which is an element of

$$\Pi_{\pi_B, \pi_C} = \{\pi \in S_{B \cup C} : b_i <_\pi b_j \text{ and } c_l < c_k \text{ for all } 1 \leq i < j \leq m \text{ and } 1 \leq l < k \leq q\},$$

where $S_{B \cup C}$ is the set of all permutations on $B \cup C = \{b_1, ..., b_m\} \cup \{c_1, ..., c_q\}$. The process underlying RIMs for generating π_{new} is as follows,

1: Initialize $\pi_{\text{new}} \leftarrow \pi_B$
2: $k \leftarrow m$
3: **for** $i = 1, ..., q$ **do**

4: Randomly select $\ell \in \{0, ..., k\}$ with probability

$$\frac{e^{-\ell\theta}}{Z_{k,q-i+1}(\theta_j)} Z_{\ell,q-i}(\theta_j)$$

5: Insert c_i into π to the left of the ℓth element of π_{new} taken from the right
6: $k \leftarrow \ell$

In the above process we iteratively insert π_C into π_B ensuring that the order of the items in π_C remains intact so $\pi_{\text{new}} \in \Pi_{\pi_B,\pi_C}$. It remains to show that the above process is well defined, that is that the probabilities in the random selection procedure sum to 1, and that after performing this process at all nodes to generate a ranking π we have $P(\pi \mid \tau(\theta)) \propto \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau))$. To show that the process is well-defined note that

$$\begin{aligned}
\sum_{\ell=0}^{k} \frac{e^{-\ell\theta_j}}{Z_{k,q-i+1}(\theta_j)} Z_{\ell,q-i}(\theta_j) &= \frac{1}{Z_{k,q-i+1}(\theta_j)} \sum_{\ell=0}^{k} e^{-\ell\theta_j} Z_{\ell,q-i}(\theta_j) \\
&= \frac{1}{Z_{k,q-i+1}(\theta_j)} Z_{k,q-i+1}(\theta_j) \\
&= 1,
\end{aligned}$$

where the second equality follows by a straightforward rephrasing of Equation (3.3.9) of Theorem 3.4 in Andrews (1976).

Now suppose we have followed the above procedure to produce the partial ranking π_{new}. Letting $m \geq j_1 \geq ... \geq j_m \geq 0$ be the indices randomly chosen during the inner loop of the above procedure and letting $j_0 = m$, note that the probability of generating $\pi_{\text{new}} \in \Pi_{\pi_B,\pi_C}$ was

$$\begin{aligned}
\prod_{i=1}^{q} \frac{e^{-j_1\theta_j}}{Z_{j_{i-1},q-i+1}(\theta_j)} Z_{j_i,q-i}(\theta_j) &= e^{-\theta_j \sum_{i=1}^{q} j_i} \prod_{i=1}^{m} \frac{Z_{j_i,q-i}(\theta_j)}{Z_{j_{i-1},q-i+1}(\theta_j)} \\
&= e^{-\theta_j \sum_{i=1}^{q} j_i} \prod_{i=1}^{q} \frac{Z_{j_i,q-i}(\theta_j)}{Z_{j_{i-1},q-i+1}(\theta_j)} \\
&= \frac{e^{-\theta_j \sum_{i=1}^{q} j_i}}{Z_{m,q}(\theta_j)}
\end{aligned}$$

where the last line follows by a telescoping argument since $Z_{r,0}(\theta_j) = 1$ for all $r \geq 0$. But $\sum_{i=1}^{q} j_i$ is exactly the number of inversions created when merging π_B, π_C together to form π_{new}. Hence it follows immediately by definitions that when we generate $\pi \in S_n$ at the root using the above generative procedure we have that

$$\begin{aligned} P(\pi \mid \tau(\theta)) &= \prod_{i_j \in \mathcal{I}} \frac{e^{-\theta_j v_j(\pi, \pi_\tau)}}{Z_{|L_j|,|R_j|}(\theta_j)} \\ &\propto \prod_{i_j \in \mathcal{I}} \exp(-\theta_j v_j(\pi, \pi_\tau)) \end{aligned}$$

as desired.

Appendix B Proof of Proposition 3.3

Note that from Equation (5) we have that, for a tree structure τ with root i_r,

$$\begin{aligned} -\frac{1}{|\mathcal{D}|} \log P(\mathcal{D} \mid \tau(\theta)) &= \sum_{i_j \in \mathcal{I}} score(j, \theta_j) \\ &= score(r, \theta_r) + \sum_{i_j \in \text{left subtree of } i_r} score(j, \theta_j) + \sum_{i_j \in \text{right subtree of } i_r} score(j, \theta_j). \end{aligned}$$

Thus we see that the log-likelihood decomposes into a score at the root node plus sums of scores at internal nodes in the right and left subtrees of the root respectively. Performing this decomposition recursively we see that the the score breaks down into scores at internal nodes plus sums of scores at their right and left subtrees respectively. This decomposition lends itself to a dynamic programming strategy for minimization.

Without loss of generality assume that $\pi_\tau = (a_1, ..., a_n)$ is the identity permutation and let $\overline{D}$ be the average discrepancy matrix of the the data as usual. For $1 \leq j < k < m \leq n$

define

$$score(j,k,m,\theta) = \theta \overline{V}_{j,k,m} + \log Z_{j-k+1,m-k}(\theta)$$

$$\theta(j,k,m) = \arg\min_{\theta} score(j,k,m,\theta)$$

where $\overline{V}_{j,k,m} = \sum_{j'=j}^{k} \sum_{m'=k+1}^{m} \overline{D}_{j'm'}$. Note that $score(j,k,m,\theta)$ is the usual score for an internal node whose left subtree has $a_j, ..., a_k$ as leaves and whose right subtree has $a_{k+1}, ..., a_m$ as leaves. Now for every $1 \leq j, m \leq n$ let $cost(j,m)$ be the negative log-likelihood of the optimal (minimum cost) tree over $a_j, ..., a_m$. Then we have the following recursive relationships

$$cost(j,j) = 0 \qquad \text{for } 1 \leq j \leq n,$$

$$cost(j,m) = \min_{j<k<m} score(j,k,m,\theta(j,k,m)) + cost(j,k) + cost(k+1,m) \qquad \text{for } 1 \leq j < m \leq n.$$

Note that $cost(1,n)$ is the negative log-likelihood of the optimal tree structure over π_τ. By keeping back pointers, $back(j,m)$, which save the value k with $j < k < m$ which minimizes $cost(j,m)$ we are able to reconstruct the ML tree structure with little additional work after recursively solving the above equations. From the above relationships the correctness of Algorithm 2 is obvious.

It remains to establish the run-time of the algorithm. Consider the non-constant time operations in the inner-most loop of the algorithm (over k). The first step in this inner loop is the calculation of $\overline{V}$. This is, naively, a $O(n^2)$ operation as it is a double sum. However it is easy to see that, on the first iteration, we will have $j - k = 1$ and thus the first iteration only requires $O(n)$ time. Moreover, on every subsequent iteration we can easily calculate the new value of $\overline{V}$ by performing $O(n)$ additions and subtractions to the old value of $\overline{V}$. Thus computing $\overline{V}$ takes only $O(n)$ time on each iteration. Next we must compute $\theta(j,k,m) = \arg\min_\theta score(j,k,m,\theta)$. As noted previously we assume this optimization terminates in a constant number of steps c. Since each gradient and score computation

requires $O(n)$ time the optimization takes $cO(n) = O(n)$ time. Finally we must compute $score(j, k, m, \theta(j, k, m))$, this also requires $O(n)$ time. Thus we see that the inner most loop of the algorithm requires $O(n)$ time on each iteration. As all other operations in the algorithm require constant time, the inner-most loop is the third nested loop, and each loop is over $\leq n$ elements, it follows that the algorithm has a $O(n^4)$ runtime.