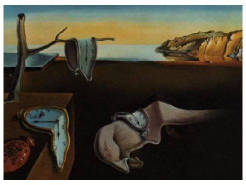

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

PerMallows: An R Package for Mallows and Generalized Mallows Models

Ekhine Irurozki **Borja Calvo** **Jose A. Lozano**
University of the Basque Country

Abstract

In this paper we present the R package **PerMallows**. The considered distances are Kendall's τ, Cayley, Hamming and Ulam and it includes functions to work with the Mallows, Generalized Mallows and Weighted Mallows models. **PerMallows** includes functions for making inference, sampling and learning such distributions, some of which are novel in the literature. As a by-product, **PerMallows** also includes operations for permutations, paying special attention in those related with the Kendall's-τ, Cayley, Ulam and Hamming distances and the random generation of permutations.

Keywords: ranking, permutation, kendall's-τ, cayley, hamming, ulam, inversion, swap, cycle, fixed point, longest increasing subsequence, mallows, generalized mallows, weighted mallows, learning, sampling, R .

1. Introduction

Permutations are ordered sets of items that arise naturally in many domains, such as genomics (Bader 2011), cryptography, scheduling or computer vision (Ziegler, Christiansen, Kriegman, and Belongie 2012). One of the hottest topics is preference learning (Furnkranz and Hullermeier 2013), since its commercial applications have increased exponentially in the last years.

The particularities of permutation data does not allow us to straightforwardly adapt the methodology used over other domains to the permutations domain. Consider, for example, the notion of independence. For a distribution over binary variables, independence between two variables implies that the first variable will have a value or 0 or 1 regardless of the value of the second variable. However, any two given positions of a permutation can not have the same value by any chance, so the classic notion of independence does not naturally translate into the permutations domain.

The conclusion is that permutation spaces need specific probability models and concepts. However, the requirements for these models are very much the same, that is, the ability to accurately model real data and the efficient computation of the most common operations, such as inference, learning and sampling. In order for these operations to be efficient, the particular nature of permutations must be considered. In other words, efficient algorithms for dealing with permutations and its common operations are the key to efficient algorithms for probability models over permutations.

We have implemented two of the most popular probability models on permutations, the Mallows model (MM) (Mallows 1957) and the Generalized Mallows model (GMM) (Fligner and Verducci 1986) and an adaptation of the GMM which goes by the name of Weighted Mallows model (WMM) (Irurozki, Calvo, and Lozano 2014b). They all are exponential models and rely on a distance for permutations. The MM has been used to define a family of models that considers any of the following distances: Kendall's-τ, Cayley, Ulam, Hamming, Spearman's-ρ and Spearman's footrule. This family is referred to as distance-based models (Diaconis 1988). Moreover, when the distance for permutations is Kendall's-τ the MM is also known as the Mallows ϕ-model (Critchlow, Fligner, and Verducci 1991).

The GMM is the most referenced among the extensions of the MM. The GMM under the Kendall's-τ distance is also known as Mallows ϕ-component model. Like the MM, the GMM is also a unimodal distribution. However, instead of one single spread parameter, GMM makes use of $n-1$ dispersion parameters. The idea is to model situations in which the distribution is close to uniformity for some positions of the permutation while being close to the mode for some others.

The literature includes many theoretical discussion references (Fligner and Verducci 1988), (Critchlow *et al.* 1991), (Fligner and Verducci 1986) as the analysis of the asymptotic properties of, for example, the length of the longest increasing subsequence (LIS) (Mueller and Starr 2013), (Féray 2013). Moreover, its practical relevance is supported by the application papers in different disciplines such as Preference Elicitation (Chen and Pu 2004), Information Retrieval (Farah and Vanderpooten 2007), Classification (Cheng and Hullermeier 2009), Label Ranking (Cheng and Hüllermeier 2009) or Evolutionary Algorithms (Ceberio, Mendiburu, and Lozano 2011). The models can be easily extended and adapted, as done for non-parametric models (Mao and Lebanon 2008), infinite permutations (Gnedin and Olshanski 2012), (Meila and Bao 2008) and mixture models (D'Elia and Piccolo 2005), (Meila and Chen 2010), (Murphy and Martin 2003), (Lee and Yu 2012) signed permutations (Arora and Meila 2013)

The four metrics for permutations considered in the **PerMallows** package are Kendall's-τ, Cayley, Hamming and Ulam. Each of these distances reflects a different nature: The Kendall's-τ distance is related with the number of inversions of a permutation, Cayley is related with the number of swaps and also with the cyclic structure of permutations, Hamming is related with the number of fixed points and Ulam with the longest common subsequence between two permutations.

This is not the first package in the literature to deal with distributions over permutations. The **prefmod** package (Hatzinger and Dittrich 2012) models preference data in the form of paired comparisons. In the particular case of the distance-based models, there exist two packages. The **RMallow** package uses a EM algorithm to fit mixtures of MM under the Kendall's-τ distance to full or partial rankings, with and without ties as described in Murphy and Martin (2003). The **pmr** (Probability Models for Ranking Data) (Lee and Yu 2014) on the other hand, considers Luce's model (Critchlow *et al.* 1991), MM and GMM under the Kendall's-τ, Spearman's-ρ, Spearman's-ρ^2 and foot-rule distances. The package is aimed at helping in the analysis of preference data in tasks such as visualizing data and computing descriptive statistics. It implements functions for the maximum likelihood estimation for the parameters of the MM and extensions thereof similar to the GMM.

None of the aforementioned packages offer a wide range of functionalities so as to be considered a complete toolbox for reasoning on permutation data. There is no way of generating permutations from a given model or calculating the probability of a permutation under a certain model for example. Moreover, none of the packages consider the algebraic machinery necessary for the efficient management of functions over permutations spaces, from basic operations such as compositions or factorization of permutations into disjoint cycles to complex combinatorial functions such as counting or generating permutations.

Regarding the distance metrics, in the packages in the literature dealing with distance-based models only Spearman's-ρ, Spearman's-ρ^2 Spearman's-foot-rule and Kendall's-τ have been considered, which are the most natural for the preference domain. However, the application of permutations is beyond the preference or voting domain and the Cayley, Hamming or Ulam metrics are more natural in fields such as computer vision, biology, cryptography, matching or card shuffling.

PerMallows, a freely available R package, aims to be a compilation of functions for working on distance-based probability models MM and GMM under the Kendall's-τ, Cayley, Hamming and Ulam. The utilities include the following functions:

- Functions for dealing with MM and extensions thereof: learning the parameters from a collection of permutations, sampling permutations from a given distribution, computation of the expectation, marginal probabilities and computation of the probability of a permutation. Several algorithms for each of the tasks are offered, including approximate and exact algorithms.

- Distance related functions: compute distances between permutations, generate permutations at random at a given distance, count the number of permutations at a given distance, etc.

- Operations with permutations: generation of all permutations of a given number of items, inversion, composition, different operators such as, for example, swapping or transposing items, descriptive statistics, factorization of permutations, etc.

This paper is organized as follows. Section 2 comprises the basic definitions and results related with permutations and probability models. In particular, Section 2.1 introduces the notation and basic concepts on permutations, several operations for dealing with this particular data type are also introduced and explained. The probability models, MM and GMM, are detailed in Section 2.2 while sections 2.3 and 2.4 introduce several algorithms for sampling and learning respectively. The usage of the **PerMallows** package is shown in Section 3. Section 4 concludes the paper.

2. Dealing with permutations

In this section we give the theoretical background for the understanding of the functions in the **PerMallows** package. It is divided into four parts. First, the notions over permutations are given. Then, we briefly describe MM and GMM. The last two parts are devoted to the learning and sampling algorithms for the MM and GMM included in **PerMallows**.

2.1. Permutations and metrics

Permutations are bijections of the set of integers $\{1, \ldots, n\}$ onto itself. We will denote permutations with Greek letters, mostly π and σ. In the permutation $\sigma = [2413]$ we will say that item 2 is at position 1 and denote it $\sigma(1) = 2$. The permutation that places every item i at position i is called the identity permutation and it is denoted as $e = [123 \ldots n]$.

For every permutation σ, its inverse is denoted as σ^{-1} and defined as $\sigma^{-1}(i) = j \Leftrightarrow \sigma(j) = i$. Two permutations can be composed resulting in a new permutation. The composition operation, denoted $\sigma \circ \pi$ or $\sigma\pi$, is defined as $\sigma\pi(i) = \sigma(\pi(i))$. It is worth noticing that $\sigma\sigma^{-1} = e$.

There are many distance metrics for permutations. Four are considered in **PerMallows**, Kendall's-τ, Cayley, Ulam and Hamming. All these metrics are right invariant, meaning that for every permutation τ, $d(\sigma, \pi) = d(\sigma\tau, \pi\tau)$. In particular taking $\tau = \pi^{-1}$ and since $\pi\pi^{-1} = e$, one can w.l.o.g. write $d(\sigma, \pi) = d(\sigma\pi^{-1}, e)$. The right invariant property implies that we can always take the identity permutation as the reference. For the sake of clarity, the distance is denoted as a one parameter function when the reference permutation is the identity, $d(\sigma\pi^{-1}, e) = d(\sigma\pi^{-1})$ which simplifies the notation.

Kendall's-τ The Kendall's-τ distance $d_k(\sigma\pi^{-1})$ counts the number of pairwise disagreements between σ and π, i.e., the number of item pairs that have a relative order in one permutation and a different order in the other. We can equivalently define $d_k(\sigma\pi^{-1})$ as the number of adjacent swaps to convert σ^{-1} into π^{-1}. The maximum value of the Kendall's-τ distance between two permutations is $n(n-1)/2$. It is used mainly in voting theory and it is sometimes called bubble sort distance because $d_k(\sigma)$ equals the number of adjacent swaps that the bubble sort algorithm performs to order the items in σ increasingly. The Kendall's-τ distance can be computed in time $O(n^2)$.

A distance decomposition vector $V(\sigma) = (V_1(\sigma), \ldots, V_{n-1}(\sigma))$ can be defined such that $d_k(\sigma) = \sum_{j=1}^{n-1} V_j(\sigma)$ where $V_j(\sigma)$ equals the number of times that the bubble sort algorithm swaps item $\sigma(j)$. It follows that $0 \leq V_j(\sigma) \leq n - j$ for $1 \leq j \leq n$. Note that $V_j(\sigma)$ is also equal to the number of items smaller than $\sigma(j)$ in the tail of the permutation, and that it can be expressed as follows:

$$V_j(\sigma) = \sum_{i=j+1}^{n} I(\sigma(i) < \sigma(j)) \tag{1}$$

where $I(\cdot)$ denotes the indicator function. It is worth noticing that there is a bijection between each $\sigma \in S_n$ and each possible $V(\sigma)$ vector. Therefore, when dealing with the Kendall's-τ distance we can use the $V(\sigma)$ vector as an alternative representation of σ. The conversion from $V(\sigma)$ to σ and vice versa is supported in **PerMallows** and done in time $O(n^2)$.

Cayley The Cayley distance $d_c(\sigma\pi^{-1})$ counts the number of swaps (not necessarily adjacent) that have to be made to transform σ into π. The maximum value of the Cayley distance between two permutations is $n-1$. When the reference permutation is the identity, $d_c(\sigma)$ equals n minus the number of cycles of σ. Recall that for every $1 \leq i, j \leq n$ such that $\sigma(i) = j$, then i and j are in the same cycle, and therefore, $\sigma = [156423]$ is written in cycle notation as $(1)(2,5)(3,6)(4)$. It can be computed in time $O(n)$. Cayley distance $d_c(\sigma)$ can be decomposed into a vector $X(\sigma)$ of $n-1$ binary terms, $X(\sigma) = (X_1(\sigma), \ldots, X_{n-1}(\sigma))$ where $X_j(\sigma) = 0$ iff j is the largest item in its cycle in σ. It follows from the definition that $d_c(\sigma) = \sum_{j=1}^{n-1} X_j(\sigma)$. Contrary to the Kendall's-τ distance, there is not a bijection between every possible $X(\sigma)$ vector and $\sigma \in S_n$: although each $\sigma \in S_n$ has one unique decomposition vector $X(\sigma)$, the opposite is not necessarily true. (Irurozki, Calvo, and Lozano 2014a) provides two algorithms that given $X(\sigma)$, randomly generate a permutation σ consistent with $X(\sigma)$ and assuming that every permutation consistent with $X(\sigma)$ is equally probable. The conversions from a permutation σ to $X(\sigma)$ is also supported in **PerMallows**. The three conversions have time complexity $O(n)$.

Hamming The Hamming distance $d_h(\sigma\pi^{-1})$ counts the number of point-wise disagreements between two permutations. The maximum value of the Hamming distance between two permutations is, therefore, n. It is worth noticing that there is no pair of permutations σ and π such that $d(\sigma, \pi) = 1$. The Hamming distance is closely related to the concepts of fixed and unfixed points. A fixed point in σ is a position i where $\sigma(i) = i$ while the opposite is an unfixed point. The relation between the Hamming distance and the fixed points comes from the fact that the Hamming distance to the identity, $d_h(\sigma)$, counts the number of unfixed points in σ. It can be computed in time $O(n)$.

The notion of fixed points leads to the decomposition vector of the Hamming distance, $H(\sigma) = (H_1(\sigma), \ldots, H_n(\sigma))$ where $H_j(\sigma) = 0$ iff j is a fixed point in σ. Consequently, $d_h(\sigma) = \sum_{j=1}^{n} H_j(\sigma)$. Note that every $\sigma \in S_n$ has a unique $H(\sigma)$ but the opposite is not necessarily true. The conversion from σ to $H(\sigma)$ and the generation uniformly at random of σ consistent with a given $H(\sigma)$ are both supported in **PerMallows** and have complexity $O(n)$.

Ulam The Ulam distance $d_u(\sigma\pi^{-1})$ counts the length of the complement of the longest common subsequence (LCS) in σ and π, i.e., the number of items which are not part of the LCS. The maximum value of the Ulam distance between two permutations is $n-1$. If the reference permutation is the identity, $d_u(\sigma)$ equals n minus the length of the longest increasing subsequence (LIS). The classical example to illustrate the Ulam distance $d_u(\sigma, \pi)$ considers a shelf of books in the order specified by σ (Diaconis 1988). The objective is to order the books as specified by π with the minimum possible number of movements, where a movement consists of taking a book and inserting it in another position (delete-insert). The minimum number of movements is exactly $d_u(\sigma, \pi)$.

A permutation can possibly have several LIS, as for example $\sigma = [7615243]$, where there are two LIS of length 3, one consisting of items 1, 2 and 4 and the other one consisting of items 1, 2 and 3.

The complexity of computing the Ulam distance between two given permutations is $O(n \, log \, l)$ where l is the length of the longest increasing subsequence.

Counting and generating permutations

The random generation of permutations is a problem of interest in many disciplines. The uniformly at random generation, for example, can be efficiently carried out with the well known Fisher-Yates shuffle (also known as Knuth shuffle). A more restrictive version of the problem is that of generating a permutation at distance d from the identity. This problem is closely related to that of counting the number of permutations at distance d from the identity. In this section we show how to solve these two problems, that is:

- Given the number of items n and a distance d, how many permutations are there at distance d from the identity, e?
- Given the number of items n and a distance d, generate uniformily at random a permutation at the given distance from e.

Clearly, the number of permutations depends on whether the distance is Kendall's-τ, Cayley, Hamming or Ulam. The question of counting the number of permutations at distance d, denoted $S(n, d)$, has been largely studied for several distances. There is no closed expression for $S(n, d)$ for any of the metrics in this paper. Fortunately, these sequences appear in the Online encyclopedia of Integer Sequences (OEIS)[1] for every metric considered in **PerMallows** for different values of n and d, as well as some recursions to obtain them.

Kendall's-τ The number of permutations at each possible Kendall's-τ distance, $S_k(n, d)$, is the sequence with code A008302 in the OEIS. Its computational cost is $O(n \cdot d_{max}) \equiv O(n^3)$.

The next question is the uniformly at random generation of a permutation at distance d from the identity. Note that this is equivalent to generating a permutation with a given number of inversions, which, until now, was an open problem (Arndt 2010). We now propose a novel algorithm to solve this question.

Recall that there is a bijection between every permutation σ of n items and every possible $V(\sigma)$ decomposition vector. Every position $1 \leq j < n$ of vector $V(\sigma)$, $V_j(\sigma)$, is restricted to having values $0 \leq (n - j)$. Moreover, $d_k(\sigma) = \sum_{j=1}^{n-1} V_j(\sigma)$. The problem of randomly generating a permutation at distance d from the identity is thus equivalent to that of randomly generating a vector $V(\sigma)$ such that $0 \leq V_j(\sigma) \leq (n - j)$ for $1 \leq j < n$. This induces the next alternative representation for the problem: Given $n - 1$ buckets of capacity $n - 1, n - 2, n - 3, \ldots, 1$, distribute d indistinguishable balls in such a way that any possible configuration is equally probable.

Since the number of permutations of n items at Kendall's-τ distance d is $S_k(n, d)$, then, the number of $V(\sigma)$ vectors such that $\sum_{j=1}^{n-1} V_j(\sigma) = d$ is $S_k(n, d)$. Equivalently, the number of ways to distribute d indistinguishable balls into $n-1$ buckets of capacity $n-1, n-2, n-3, \ldots, 1$ is $S_k(n, d)$.

We approach this problem with a recursive procedure. Its base case is $d \leq 0$ where nothing is done. In the general case, we introduce k balls in the first bucket and then recursively solve the problem with the rest of the $n - 2$ buckets and $d - k$ balls. The main question is how to choose k if we want every configuration to be equally probable. The insertion of k balls in the first bucket implies that there will be $S_k(n - 1, d - k)$ possible ways to introduce the

[1] http://oeis.org/

$d-k$ remaining balls into the $n-2$ remaining buckets. In other words, of the $S_k(n,d)$ total possible configurations of distributing d balls in the $n-1$ buckets, exactly $S_k(n-1,d-k)$ of them will have k balls in the first bucket. Therefore, for a u.a.r. distribution of the balls into the buckets, the probability of introducing k balls in the first bucket is

$$\frac{S_k(n-1,d-k)}{S_k(n,d)}$$

The analogy with the generation of the $V(\sigma)$ vector is quite trivial. Instead of the $n-1$ buckets of capacity $n-1, n-2, n-3, \ldots, 1$, we have a vector $V(\sigma)$ of $n-1$ positions, each restricted to have values smaller than or equal to $n-1, n-2, n-3, \ldots, 1$. The base case is the situation when $d \leq 0$. In this case the output and input vectors are the same. Otherwise, we randomly choose k, where the probability of setting $P(V_1(\sigma) = k)$ equals $S_k(n-1,d-k)/S_k(n,d)$. Then, we recursively solve the problem of inserting $d-k$ balls into the remaining positions of the vector, $V_2(\sigma), \ldots, V_{n-1}(\sigma)$.

Regarding the computational complexity of the current algorithm, if the values for $S_k(n,d)$ are given, the cost of generating a permutation at Kendall's-τ distance d is $O(n^2)$.

Cayley Recall that the Cayley distance $d_c(\sigma)$ equals n minus the number of cycles of σ. Therefore, the number of permutations at Cayley distance d, $S_c(n,d)$, equals the number of permutations with $k = n-d$ cycles. The number of permutations of n items with k cycles is given by the Stirling numbers of the first kind, which has code A008275 in the OEIS. The time complexity of computing the Stirling numbers is $O(n \cdot d_{max}) \equiv O(n^2)$.

PerMallows also includes a recursive process for the u.a.r. generation of a permutation at Cayley distance d. Briefly, the generation of a permutation of n items and k cycles implies firstly, the recursive generation of a permutation of the first $n-1$ items and then, the insertion of item n. This is due to the fact that the whole set of permutations of n items and k cycles can be split into two groups: The group of permutations in which item n is in a cycle of length one, i.e. $\sigma(n) \equiv n$, and the group of permutations in which item n is in a cycle of length greater than one ($\sigma(n) \neq n$). If the cardinality of those sets can be given, the generation of the permutation can be efficiently performed.

The computational cost of generating a random permutation at Cayley distance d from the identity given the Stirling numbers is $O(n)$. Details of the procedure can be found in Irurozki *et al.* (2014a).

Hamming The question of counting the number of permutations at Hamming distance d, $S_h(n,d)$ is closely related to the notion of derangement, that is, a permutation with no fixed point. Recall that, in a permutation at Hamming distance d there are d unfixed points, i.e., there are d items that form a derangement. Therefore, the number of permutations of n items at Hamming distance d is

$$S_h(n,d) = \binom{n}{d} S(d) \quad \text{where } S(d) \text{ is the number of derangements of } d \text{ items}$$

The number of derangements of n items is the sequence with code A000166 in the OEIS. Even though there is no closed form for $S(d)$, the time complexity for the computation of $S(d)$ is $O(d)$, so $S_h(n,d)$ has time complexity $O(n)$.

PerMallows also includes a recursive process for the u.a.r. generation of a permutation of n items at Hamming distance d. It is based on the random generation of a derangement of d items, which is a well known problem (Akl 1980). The details of this process can be found in Irurozki *et al.* (2014b).

Ulam Algorithms for counting $S_u(n, d)$ and for the u.a.r. generation of permutations of n items at Ulam distance d are included in **PerMallows**. The sequence $S_u(n, d)$ appears in the OEIS with code A126065.

Both processes of counting and generating permutations at a given Ulam distance are related with the notions of Ferrers diagrams (FD) and Standard Young Tableaux (SYT). The link between SYT and permutations is given by the celebrated Robinson-Schensted-Knuth (RSK) correspondence. A self contained explanation of the computation of $S_u(n, d)$ and the random generation of permutations at a given Ulam distance can be found in Irurozki, Calvo, and Lozano (2014c). The complexity of the counting and generating processes is equivalent to the complexity of enumerating the partitions of n, which has not got any closed form.

Counting permutations at a given distance is a crucial operation, not only for the random generation of permutations, but also for the learning and sampling processes. Among the distances considered, Ulam is the most expensive from a computational perspective. If a user needs to work usually with with permutations of a particular n (for large n, say $n > 50$), **PerMallows** includes functions to speed the computation up. The idea is to generate files containing relevant information such as the counts of the permutations at each possible distance.

2.2. Probability distributions over permutations

This section introduces the Mallows model (Mallows 1957) and its most popular extension, the Generalized Mallows model (Fligner and Verducci 1986).

Mallows model

The Mallows model (MM) was one of first probability models proposed for rankings or permutations. However, it is still one of the most used models in both theoretical and application papers. It is an exponential model defined by the central permutation σ_0 and the spread (or dispersion) parameter θ. When $\theta > 0$, σ_0 is the mode of the distribution, i.e., the permutation with the highest probability. The probability of any other permutation decays exponentially as its distance to the central permutation increases. The spread parameter controls how fast this fall happens. It can be expressed as follows:

$$p(\sigma) = \frac{exp(-\theta d(\sigma\sigma_0^{-1}))}{\psi(\theta)}$$

where $\psi(\theta)$ is the normalization constant. It is considered as the analogous distribution of the Gaussian distribution for distributions over permutations. Note that when the dispersion parameter θ is greater than 0, then σ_0 is the mode. On the other hand, with $\theta = 0$ we obtain the uniform distribution and when $\theta < 0$ then σ_0 is the anti-mode, i.e., the permutation with the lowest probability.

Generalized Mallows model

This extension of the MM tries to break the restriction that imposes every permutation at the same distance to have the same probability value. Instead of one single spread parameter, the generalized Mallows model (GMM) requires the definition of $n-1$ spread parameters θ_j for $1 \leq j < n$, each affecting a particular position of the permutation. This allows modeling a distribution with more emphasis on the consensus of certain positions of the permutation while having more uncertainty about some others. Not every distance that can be used in the MM can also be used in GMM, since GMM requires the distance to be decomposed in $n-1$ terms as follows:

$$d(\sigma\sigma_0^{-1}) = \sum_{j=1}^{n-1} S_j(\sigma\sigma_0^{-1}) \tag{2}$$

For any distance that decomposes as the above equation, the GMM is defined as follows:

$$p(\sigma) = \frac{exp(\sum_{j=1}^{n-1} -\theta_j S_j(\sigma\sigma_0^{-1}))}{\psi(\boldsymbol{\theta})}$$

where the dispersion parameter $\boldsymbol{\theta} = (\theta_1, \dots, \theta_{n-1})$ is multidimensional and $\psi(\boldsymbol{\theta})$ is the normalization constant.

Normalization constant under each distance

This section is devoted to the efficient computation of the normalization constants $\psi(\theta)$ (resp. $\psi(\boldsymbol{\theta})$) of the MM (resp. GMM). The distribution and the normalization constants can be factorized under Kendall's-τ and Cayley distances but there is no known closed expression for the normalization constant of the models under Hamming or Ulam. In these cases, the naive computation consists of summing $n!$ terms, so it is infeasible for medium size values of n. In this section we show efficient formulas for $\psi(\theta)$ and $\psi(\boldsymbol{\theta})$ and particular properties of both MM and GMM under each of the metrics considered are stated.

Kendall's-τ The Kendall's-τ distance can be used with the GMM since $d_k(\sigma)$ can be decomposed (see Equation (1)). Moreover, $\psi(\boldsymbol{\theta})$ has a closed expression under the Kendall's-τ distance. The GMM under the Kendall's-τ distance can be expressed as follows (Fligner and Verducci 1986):

$$p(\sigma) = \prod_{j=1}^{n-1} \frac{exp(-\theta_j V_j(\sigma\sigma_0^{-1}))}{\psi_j(\theta_j)} \qquad \text{where } \psi_j(\theta_j) = \frac{1 - exp(-\theta_j(n-j+1))}{1 - exp(-\theta_j)} \tag{3}$$

Since the MM is the particular case of the GMM where every θ_j has equal value, this expression can also be used for the MM under the Kendall's-τ distance.

Cayley Cayley distance can be decomposed as in Equation (2) as shown in Section 2.1. Therefore, the GMM under the Cayley distance can be factorized and, as a consequence,

$\psi(\boldsymbol{\theta}) = \prod_{j=1}^{n-1} \psi_j(\theta_j)$. Formally, the GMM under the Cayley distance can be written as follows:

$$p(\sigma) = \prod_{j=1}^{n-1} \frac{exp(-\theta_j X_j(\sigma\sigma_0^{-1}))}{\psi_j(\theta_j)} \qquad \text{where } \psi_j(\theta_j) = (n-j)exp(-\theta_j) + 1 \tag{4}$$

Although the factorization was introduced in Fligner and Verducci (1986), we correct here some typos of the original paper. Since the MM is the particular case of the GMM where every θ_j has equal value, this expression can also be used for the MM under the Cayley distance.

Hamming The closed form of $\psi(\theta)$ under MM has already been given in Fligner and Verducci (1986). An alternative expression and its proof can be found in Irurozki *et al.* (2014b). The normalization constant of the MM under the Hamming distance is as follows:

$$\psi(\theta) = n!\, exp(-\theta n) \sum_{k=0}^{n} \frac{(exp(\theta) - 1)^k}{k!}$$

The Hamming distance can not be decomposed as in Equation (2). This means that there is no GMM under the Hamming distance. Nevertheless, the idea behind the GMM can be adapted for the Hamming distance defining a different dispersion parameter, θ_j, $1 \leq j \leq n$, for each position of the permutation (Irurozki *et al.* 2014b). This version has been called Weighted Mallows model (WMM) under the Hamming distance and the normalization constant can be efficiently computed as follows:

$$\psi(\boldsymbol{\theta}) = exp(-\sum_j \theta_j) \sum_{k=0}^{n} (n-k)!\gamma_k(exp(-\theta_1) - 1, \ldots, exp(-\theta_n) - 1)$$

where γ_k is the Elementary Symmetric Polynomial of degree k (Irurozki *et al.* 2014b).

Both GMM and WMM allow modeling situations where there is a different consensus for different positions of the permutation. The most remarkable difference among GMM and WMM is that the former uses a vector of $n-1$ dispersion parameters and the latter uses n dispersion parameters. However, for the sake of clarity and given their similarities, we will denote the WMM as a GMM.

Ulam The Ulam distance has not got a natural decomposition of the form of Equation (2) and, thus, the GMM cannot be defined under this metric. Moreover, for the MM there is not a closed form for the normalization constant $\psi(\theta)$. However, an efficient way of computing $\psi(\theta)$ can be found in Irurozki *et al.* (2014c). It is based on the fact that every permutation at the same distance has equal probability. Considering this fact, the normalization constant $\psi(\theta) = \sum_\sigma exp(-\theta d_u(\sigma\sigma_0^{-1}))$ can be expressed as the sum of n terms in the following way:

$$\psi(\theta) = \sum_{d=0}^{n-1} S_u(n,d) exp(-\theta d) \tag{5}$$

This expression relies on the computation of the number of permutations of n items at each possible Ulam distance, $S_u(n,d)$.

2.3. Sampling

PerMallows implements three different algorithms for generating permutations from a given distribution. The Distances algorithm can generate samples from the MM, while the Multistage and Gibbs can generate samples from both MM and GMM.

Distances sampling algorithm

The Distances sampling algorithm can generate samples from the MM under the Kendall's-τ, Cayley (Irurozki *et al.* 2014a), Hamming (Irurozki *et al.* 2014b) and Ulam (Irurozki *et al.* 2014c) metrics. It is based on the fact that every permutation at the same distance from the central permutation has the same probability. Therefore, it can be used to sample from the MM but not for the GMM.

This sampling method proceeds as follows. Let the metric $d(\sigma\sigma_0^{-1})$ range between 0 and d_{max} for permutations of n items. Also, let $S(n,d)$ be the number of permutations at distance d from the identity permutation[2]. Then, the probability of obtaining a permutation at distance d is as follows:

$$p(\sigma|d(\sigma,\sigma_0)=d) = \frac{S(n,d)exp(-\theta d)}{\psi(\theta)} \tag{6}$$

Note that the normalization constant $\psi(\theta) = \sum_\sigma exp(-\theta d(\sigma\sigma_0^{-1}))$ can be expressed as the sum of d_{max} terms in the following way:

$$\psi(\theta) = \sum_{d=0}^{d_{max}} S(n,d)exp(-\theta d)$$

Taking into consideration the previous expressions of the probability function, the process of simulating from the distribution can be carried out in three stages:

1. Randomly select the distance at which the permutation will lie using Equation (6).
2. Pick uniformly at random a permutation π at distance d from the identity permutation e, i.e. $d(\pi) = d$. This step relies on the u.a.r. generation of a permutation at a given distance.
3. In case $\sigma_0 = e$, then π is output. Otherwise, the invariance property lets us obtain $\sigma = \pi\sigma_0$, since $d = d(\pi) = d(\pi\sigma_0, \sigma_0) = d(\sigma, \sigma_0)$.

The complexity of computing the probabilities of step 1 is equivalent to the complexity of computing the number permutations per distance and has to be computed once per run. The second step is as complex as generating a permutation at a given distance. The complexities depend on the considered distance, see Section 2.1.1 for their explicit expression.

We can conclude that this is a quick as well as precise algorithm for the simulation of the MM. However, it has some limitations. First, it does not work with the GMM. Also, it is infeasible to keep $S(n,d)$ for values of $n > 150$. In these situations we can use the Multistage sampling algorithm.

The Ulam version of the Distances sampling algorithm is the most expensive from a computational perspective. We have stated that **PerMallows** is capable of generating auxiliary files

[2]Due to the invariance property, every permutation has the same number of permutations at distance d.

for accelerating part of the computation. These files can be useful for the Distances sampling process.

Multistage sampling algorithm

The Multistage sampling algorithm generates samples from the MM and GMM under the Kendall's-τ, Cayley and Hamming metrics. It is based on randomly generating a decomposition vector (see Section 2.1) from the MM or GMM

This algorithm divides the sampling process into three stages, namely:

1. Randomly generate a distance decomposition vector. For the Kendall's-τ and Cayley distances this vector is sampled using the probability of each of the terms of the decomposition vector (Fligner and Verducci 1986), (Irurozki *et al.* 2014a). For the Hamming distance, the decomposition vector is generated considering the marginal probabilities as shown in Irurozki *et al.* (2014b), where this algorithm is known as the Chain sampling algorithm.

2. Generate a permutation uniformly at random π consistent with the given distance decomposition vector.

3. In case $\sigma_0 = e$, then π is output. Otherwise, we get the final permutation by composing π with σ_0, obtaining $\sigma = \pi\sigma_0$ since $\pi = \sigma_0\sigma_0^{-1}$.

The complexity of step 1 is $O(n)$ for the Kendall's-τ and Cayley and $O(n^2)$ for Hamming. The complexity of step 2 is detailed in Section 2.1. This method is not as fast as the Distances; however, it can generate permutations from both MM and GMM and it can efficiently handle distributions on permutations of large n.

Gibbs sampling algorithm

The Gibbs sampler is a Markov Chain Monte Carlo algorithm based on sampling a Markov chain whose stationary distribution is the distribution of interest. Therefore, we have adapted this algorithm to generate samples from approximate distributions of both MM and WMM.

The Gibbs sampler generates permutations by moving from one solution to another. From a particular permutation, it is possible to move to any other permutation if and only if it is in its neighborhood. We have considered the following neighborhoods. Under the Kendall's-τ distance, the neighborhood of a permutation is the set of permutations that result from randomly selecting a position $1 \leq i < n$ and swapping $\sigma(i)$ and $\sigma(i+1)$. On the other hand, under the Cayley and Hamming distances, the neighborhood of a permutation is the set of permutations that result of randomly selecting two positions $1 \leq i < j \leq n$ and swapping $\sigma(i)$ and $\sigma(j)$. Finally, under the Ulam distance, the neighborhood consists of the set of permutations that result from deleting an item from its position and inserting it in any other position, shifting other items between them.

The Gibbs algorithm proceeds as follows:

1. Generate uniformly at random a permutation σ.

2. Select uniformly at random a permutation σ' in the neighborhood of σ.

3. Let $\gamma = min\{1, p(\sigma')/p(\sigma)\}$. With probability γ the algorithm accepts the candidate permutation moving the chain to the candidate permutation, $\sigma = \sigma'$, and goes back to step 2. Otherwise, it discards σ' and goes back to step 2.

The initial samples are discarded (burn-in period) until the Markov chain approaches its stationary distribution and so samples from the chain are samples from the distribution of interest. Then, the above process is repeated until the algorithm generates a given number of permutations. Recall that so far we have assumed that the central permutation is the identity, $e = [123 \dots n]$. If not, we can center the sample around σ_0 by composing each of the permutations σ in the sample with σ_0.

The Gibbs algorithm is the fastest of the three algorithms considered. However, we should emphasize the fact that this an approximate sampling algorithm.

2.4. Learning

In this section we deal with the maximum likelihood estimation of the parameters of the distribution given a sample of m i.i.d. permutations $\{\sigma_1, \sigma_2, \dots, \sigma_m\}$ which is given by the following equation.

$$Ln\ \mathcal{L}(\{\sigma_1, \sigma_2, \dots, \sigma_m\}|\sigma_0, \boldsymbol{\theta}) = \sum_{s=1}^{m} Ln\ p(\sigma_s|\sigma_0, \boldsymbol{\theta})$$

The maximum likelihood parameters are different for MM and GMM. Moreover, their expression differs regarding the distance on the permutations considered. In this way, we will describe the maximum likelihood estimation for each model and distance separately. Finally, we will introduce the algorithms provided in **PerMallows** to fit the parameters of a given sample.

Mallows model

For the MM, the maximum likelihood estimation of the parameters can be carried out in two stages. The first one deals with the MLE for the central permutation, $\hat{\sigma}_0$, and then the MLE for the dispersion parameters for the given $\hat{\sigma}_0$ can be obtained.

Problems consisting of finding the permutation that minimizes the sum of the distances to at least three given permutations are called median problems. The median problem for the Kendall's-τ distance is also known as Kemeny rank aggregation problem and it is known to be NP-hard (Bartholdi, Tovey, and Trick 1989). The median problem for the Hamming distance is equivalent to the Linear Assignment Problem (LAP), for which a polynomial algorithm is known. Though the complexity of the median problems under the Cayley and Ulam distances has not been yet established, they are supposed to be NP-complete (Popov 2007).

Once the MLE for σ_0 has been found, the process continues by estimating the MLE for the dispersion parameter θ, which will also differ for each distance. When the metric is the Kendall's-τ, the MLE for the dispersion parameter, $\hat{\theta}$, is the θ that satisfies the following expression:

$$\frac{n-1}{exp(\theta)-1} - \bar{d} - \sum_{k=2}^{n} \frac{k\, exp(-\theta k)}{1-exp(-\theta k)} = 0 \qquad (7)$$

where $\bar{d} = \sum_{i=0}^{m} d_k(\sigma_i \hat{\sigma}_0^{-1})/m$. For the Cayley, Hamming and Ulam distances, the expression for the dispersion parameters can be found in Irurozki *et al.* (2014a), Irurozki *et al.* (2014b) and Irurozki *et al.* (2014c) respectively. Although there is no closed-form expression for the MLE for θ of any of them, the solution to these equations can be easily calculated with numerical methods such as Newton-Raphson. We have included in **PerMallows** both unidimensional and multidimensional Newton-Raphson from Press *et al.* (1992).

Generalized Mallows model

For the GMM, the exact maximum likelihood estimation of the parameters can not be carried out in two stages and, thus, every parameter must be simultaneously estimated. The expression of the MLE for the parameters of a GMM under the Kendall's-τ, Cayley and Hamming distances can be found in Mandhani and Meila (2009), Irurozki *et al.* (2014a) and Irurozki *et al.* (2014b) respectively.

Learning algorithms under each distance

In this section we briefly describe the learning algorithms implemented in **PerMallows** for learning MM and GMM. Since the exact estimation of the parameters as described in the previous lines can be computationally intractable under certain conditions, approximate algorithms are also included for most of the cases.

Kendall's-τ For the Kendall's-τ distance, **PerMallows** includes the approximate estimation of MM and GMM. Both estimation processes are carried out by, first, approximating the MLE for the consensus permutation $\hat{\sigma}_0$, and then obtaining $\hat{\theta}$ for the given $\hat{\sigma}_0$. The main difficulty is founded on the estimation of the consensus permutation.

The approximate estimation of the central permutations σ_0 is carried out with the Borda algorithm (Borda 1781). Borda is a heuristic algorithm that offers a very good trade-off between accuracy and time (Ali and Meila 2012). Borda builds a central ranking $\hat{\sigma}_0$ by, first, calculating the average at each position i of the permutations σ_s $1 \leq s \leq m$ in the sample, $\bar{\sigma}(i) = \sum_{s=0}^{m} \sigma_s(i)$, and secondly, sorting the items in $\hat{\sigma}_0$ according to $\bar{\sigma}(i)$. It is a fast algorithm, as it obtains a solution in time $O(nm)$. For the MM, this algorithm is asymptotically optimal as shown in Fligner and Verducci (1986).

The expression for the MLE for the dispersion parameters given a central permutation are given in Equation (7) for the MM and in Mandhani and Meila (2009) for the GMM.

The interested reader can find in Ali and Meila (2012) the description and performance analysis of several exact and approximate algorithms for the MLE for the consensus permutation under the MM. The problem of the exact MLE for the parameters of a GMM is addressed in Mandhani and Meila (2009).

Cayley **PerMallows** includes exact and heuristic algorithms for the MLE for the parameters of both MM and GMM.

The exact algorithms search the space of partial permutations of k out of n items using a branch and bound strategy. Note that the exact algorithm for the MM looks for the permutation that minimizes the sum of the distances to the permutations in the sample while the algorithm for the MLE for the parameters of the GMM looks for the permutation that maximizes the likelihood. These exact algorithms are capable of evaluating the first k positions of the permutation and also giving a lower bound in the evaluation of the rest of the permutation. These two features combined are used for bounding the search space and give rise to an efficient algorithm when there is some consensus in the sample.

The approximate algorithm for the MM and GMM is divided into two separate stages. The first one constructs an approximate MLE for the consensus permutation which will be used as an initial solution to the Variable Neighborhood Search (VNS) in the second stage. The VNS is a heuristic algorithm that searches the space of permutations improving the likelihood. It makes use of two separate neighborhood systems, namely the insert and the swap. The insert operator considers as neighbors all those permutations that result from inserting an item into another position. The swap operator considers as neighbors all those permutations that result from swapping two positions. The local search for both neighborhood systems selects at each iteration the best neighbor of the current one (or selects a solution uniformly at random among those with the best evaluation in case of ties). We refer the interested reader to Irurozki *et al.* (2014a) for further details.

Hamming **PerMallows** includes an exact algorithm for the MLE for the parameters of MM and an approximate algorithm for the MLE for the parameters of a GMM.

As we have already stated, the process of calculating the MLE for the central permutation in the MM is equivalent to the Linear Assignment Problem (LAP) and can be solved in polynomial time. Therefore, the exact estimation of the central permutation is carried out by **PerMallows** with a version of the Hungarian algorithm for the resolution of the LAP. Since the exact estimation of the central ranking is computationally cheap, an approximate algorithm is not needed.

The solution to the LAP problem, σ_{LAP}, on the GMM under the Hamming distance is a consistent estimator of the central permutation. Our proposed algorithm for the estimation of the parameters selects σ_{LAP} as the central permutation and then computes the dispersion parameters by solving a system of equations with the multidimensional Newton-Raphson proposed in Press *et al.* (1992).

Both the proof of consistency and experiments showing the performance of the proposed approximation can be found in Irurozki *et al.* (2014b).

Ulam **PerMallows** supports the approximate learning of parameters of the MM.

The approximate consensus given a collection of permutations is set to be the set-median permutation. The set-median permutation is the permutation in the sample that minimizes the sum of the distances to the rest of the permutations in the sample. Note that this problem can be solved in polynomial time.

The expression of the MLE for the dispersion parameter given the consensus permutation and further details are given in Irurozki *et al.* (2014c). Recall that for the Ulam distance it is possible to generate auxiliary files with the count of the permutations at each distance. They can be helpful also in this learning process, in particular for the estimation of the dispersion parameters.

2.5. Summary

PerMallows provides functions for dealing with the Mallows and generalized Mallows models as well as a complete toolbox for working with permutations. These probability models need the definition of a metric for permutations and **PerMallows** considers four different distances: Kendall's-τ, Cayley, Hamming and Ulam. The **PerMallows** package provides three different sampling algorithms and two learning algorithms. However, not every algorithm can be applied to the models under every metric. We include in Table 1 a summary of the applicability of the algorithms for each metric.

			Kendall	Cayley	Hamming	Ulam
MM	Sampling	Distances	✓	✓	✓	✓
		Multistage	✓	✓	✓	×
		Gibbs	✓	✓	✓	×
	Learning	Exact	×	✓	✓	×
		Approximate	✓	✓	×	✓
GMM	Sampling	Distances	×	×	×	×
		Multistage	✓	✓	✓	×
		Gibbs	✓	✓	✓	×
	Learning	Exact	×	✓	✓	×
		Approximate	✓	✓	×	×

Table 1: Applicability restrictions of the algorithms for each of the supported models.

3. The PerMallows package

In this section we show how to use the **PerMallows** package. **PerMallows** includes functions to generate permutations, as well as the most common operators and distance related functions. The metrics considered are Kendall's-τ, Cayley, Hamming and Ulam. Moreover, **PerMallows** implements the algorithms for probability distributions introduced in Irurozki *et al.* (2014c,b,a). The probability models considered are Mallows (MM) and Generalized Mallows (GMM).

The number of items in the permutations, n, is denoted `perm.length` in **PerMallows**.

3.1. Permutations

Generation The most basic function consists of generating permutations. The permutations are coded as vectors of the first n natural numbers where each item appears once and once only. They can be defined by hand as follows:

```
R> sigma <- c(1, 5, 6, 4, 2, 3)
R> sigma

[1] 1 5 6 4 2 3
```

The validity of a vector as a permutation can be checked with the function `is.permutation`.

```
R> is.permutation(perm = sigma)

[1] TRUE

R> is.permutation(c(0, 1, 5, 4, 2, 3))

[1] FALSE

R> is.permutation(c(1, 8, 9, 2, 5, 3))

[1] FALSE
```

The identity permutation is that which maps every item i to position i. It can be created with the `identity.permutation` function.

```
R> identity.permutation(perm.length = 6)

[1] 1 2 3 4 5 6
```

The generation of a permutation uniformly at random is supported via the `runif.permutation` function.

```
R> runif.permutation(n = 2, perm.length = 6)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    6    5    4    3
[2,]    3    5    4    6    1    2
```

The generation of the set of every possible permutation of n items is carried out with the `permutations.of` function. Recall that the number of permutations of n items increases factorially with n and it is thus computationally expensive to generate every permutation of $n \geq 10$. By default, the `alert` argument is set to true. When `alert` is `TRUE` and n is greater than 9, an alert message is shown.

```
R> permutations.of(perm.length = 3, alert = FALSE)

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    3    2
[3,]    2    1    3
[4,]    2    3    1
[5,]    3    1    2
[6,]    3    2    1

R> permutations.of(perm.length = 10)
```

```
Do you really want to generate all  3628800  permutations? (Y/N)
[1] "Process cancelled"
```

The collection of permutations generated is stored as a 2-dimensional matrix. It is also possible to read such a matrix from disk using the function `read.permutation.file`, which also checks if every row is a valid permutation.

```
R> path = system.file("test.txt", package = "PerMallows")
R> sample = read.permutation.file(path)
```

Together with the **PerMallows** package we provide some small datasets that will be used as running examples throughout this reference manual.

```
R> data("data.1")
R> data.1

     V1 V2 V3 V4
[1,]  1  3  2  4
[2,]  2  1  3  4
[3,]  1  4  3  2
[4,]  1  2  4  3
[5,]  2  3  4  1
```

Another way of generating permutations is by ranking the ratings of a set of items. Suppose we have the results of five students in three different tests. An example of such a file is given in 'data.order'. After reading the file, we can get the ranks of each student with the system function `order.ratings`.

```
R> data("data.order")
R> data.order

   V1   V2  V3  V4   V5
1 0.1 4.20 2.0 9.4 9.00
2 6.3 2.11 0.1 5.7 4.00
3 9.0 4.50 7.1 6.3 0.21
```

```
R> order.ratings(ratings = data.order)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    2    5    4
[2,]    3    2    5    4    1
[3,]    5    2    4    3    1
```

Operations Two permutations can be composed and the result is a permutation.

```
R> sigma <- c(1, 5, 6, 4, 2, 3)
R> pi <- c(3, 5, 1, 2, 6, 4)
R> compose(perm1 = sigma, perm2 = pi)
```

```
[1] 6 2 1 5 3 4
```

Be aware that the composition is not commutative.

```
R> compose(perm1 = sigma, perm2 = pi)

[1] 6 2 1 5 3 4

R> compose(perm1 = pi, perm2 = sigma)

[1] 3 6 4 2 5 1
```

Our implementation of the composition allows one of the arguments to be a collection of permutations. In this case, every permutation in the sample is composed with the permutation in the other argument, resulting in a new collection of permutations.

```
R> tau <- c(2, 1, 3, 4)
R> compose(perm1 = data.1, perm2 = tau)

     V2 V1 V3 V4
[1,]  3  1  2  4
[2,]  1  2  3  4
[3,]  4  1  3  2
[4,]  2  1  4  3
[5,]  3  2  4  1
```

A useful summary of a sample is given by the number of permutations in the sample in which item i appears at position j. This is usually denoted as the frequency matrix or first order marginal matrix. The current package supports it via the `freq.matrix` function. The parameter `perm` can be either a sample of permutations or a single permutation. The frequency matrix of a single permutation is usually denoted permutation matrix. We will illustrate the `freq.matrix` function using the well known APA dataset.

The American Psychological Association (APA) dataset includes 15449 ballots of the election for the president in 1980. Each voter ranked at least one of the five candidates. Along with this package we distribute the 5738 ballots that ranked the five candidates by the name of `data.apa`. This dataset has been largely used in the literature. In particular, one can find in Diaconis (1989) a spectral analysis that takes into account the first order marginal of the dataset which can be computed as follows:

```
R> data("data.apa")
R> freq.matrix(perm = data.apa)

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.1835134 0.2647264 0.2288254 0.1746253 0.1483095
[2,] 0.1350645 0.1876961 0.2466016 0.2467759 0.1838620
[3,] 0.2804113 0.1673057 0.1382015 0.1829906 0.2310910
[4,] 0.2042524 0.1693970 0.1897874 0.2028581 0.2337051
[5,] 0.1967585 0.2108749 0.1965842 0.1927501 0.2030324
```

Another basic operation for permutations is inversion. The inverse of a permutation can be obtained using the function `inverse.permutation`.

```
R> inverse.permutation(perm = sigma)

[1] 1 5 6 4 2 3
```

The argument `perm` can be a single permutation or a collection of permutations. It is worth noting that the inverse of the inverse of any permutation is itself.

```
R> inverse.permutation(inverse.permutation(c(1, 4, 5, 3, 2)))

[1] 1 4 5 3 2
```

The present package also includes operators for the manipulation of permutations. The `swap` function, for example, swaps two prescribed items.

```
R> swap(perm = identity.permutation(6), i = 1, j = 3)

[1] 3 2 1 4 5 6
```

The `insert.at` function takes an item from position i and inserts it after position j.

```
R> insert.at(perm = identity.permutation(6), i = 5, j = 2)

[1] 1 2 5 3 4 6
```

An inversion at position $1 \leq i < n$ occurs when the items at positions i and $i+1$ are swapped.

```
R> inversion.at(perm = identity.permutation(6), 1)

[1] 2 1 3 4 5 6
```

Distances We will now show how to deal with functions related with the distances between permutations included in this package. These functions include the argument `dist.name` which, unless otherwise stated, is one of the following: `kendall`, `cayley`, `hamming` or `ulam`. Moreover, its default value is `kendall`.

The following computes the distance between `perm1` and `perm2` for a given metric.

```
R> pi

[1] 3 5 1 2 6 4

R> sigma

[1] 1 5 6 4 2 3
```

```
R> distance(perm1 = sigma, perm2 = pi, dist.name = "cayley")
```

```
[1] 4
```

The arguments `perm2` and `dist.name` can be omitted. In that case, `perm2` is assumed to be the identity, that means that the following code computes the Kendall's-τ distance between `pi` and e.

```
R> distance(perm1 = pi)
```

```
[1] 6
```

As we have stated, the distance from a permutation to the identity can be decomposed in a vector. Clearly, this decomposition is different regarding the metric in question. In particular, the decomposition of the Kendall's-τ distance is a vector of $n-1$ integer terms, while for the Cayley distance it is a binary vector of $n-1$ terms and for the Hamming distance it is a binary vector of n terms. These decompositions are computed by the following function:

```
R> sigma
```

```
[1] 1 5 6 4 2 3
```

```
R> v.vector <- permutation2decomposition(perm = sigma, dist.name = "kendall")
R> v.vector
```

```
[1] 0 3 3 2 0
```

```
R> x.vector <- permutation2decomposition(perm = sigma, dist.name = "cayley")
R> x.vector
```

```
[1] 0 1 1 0 0
```

```
R> h.vector <- permutation2decomposition(perm = sigma, dist.name = "hamming")
R> h.vector
```

```
[1] 0 1 1 0 1 1
```

Given that there are possibly many longest increasing subsequences in a permutation, there is no decomposition of the Ulam distance. The possible values for the `dist.name` are therefore, `kendall`, `cayley` and `hamming` and being the default value `kendall`.

The **PerMallows** package can also perform the inverse operation, that is, given a decomposition vector and a metric, obtain a permutation consistent with the vector. Recall that for Cayley and Hamming there are possibly many permutations with a particular decomposition. In these situations, the following function recovers uniformly at random one of the permutations consistent with the decomposition vector.

```
R> decomposition2permutation(vec = v.vector, dist.name = "kendall")

[1] 1 5 6 4 2 3

R> decomposition2permutation(vec = x.vector, dist.name = "cayley")

[1] 1 6 2 4 5 3

R> decomposition2permutation(vec = h.vector, dist.name = "hamming")

[1] 1 3 2 4 6 5
```

The **PerMallows** package implements a function to obtain the list of the cycles in which the permutation decomposes.

```
R> cycles = permutation2cycles(perm = sigma)
```

The `cycle2str` function can be used in order to friendly display the cycles .

```
R> cycle2str(cycles)

(1)(5 2)(6 3)(4)
```

Also, the inverse operation consisting of building a permutation given the list of cycles is supported.

```
R> cycles2permutation(cycles = cycles)

[1] 1 5 6 4 2 3
```

PerMallows includes a function to count the number of permutations of n items at distance `dist.value` for a given distance.

```
R> count.perms.distance(perm.length = 6, dist.value = 2, dist.name = "ulam")

[1] 181
```

The most computationally expensive version of any function is usually that concerning the Ulam distance. In the case where one expects to work repeatedly with a model under the Ulam distance of a particular `perm.length` number of items (being `perm.length` large, say `perm.length`$>$ 50), it is a good idea to generate auxiliary files including the count of permutations at each distance and later operating with those files. The operations that can be accelerated are counting and generating random permutations, learning and sampling. The files are generated with the following function:

```
R> generate.aux.files(perm.length = 6)
```

```
[[1]]
[1] 6
```

The fastest version of the function to count the number of permutations at a given Ulam distance is the same as the one above but the optional parameter `disk` is set to `TRUE`. In this version, the data is read from the files instead of computing it.

```
R> count.perms.distance(perm.length = 6, dist.value = 4, dist.name = "ulam",
+     disk = TRUE)

[1] 131
```

Ragarding the process of counting permutations under the Cayley distance, we also include a function to count the number of permutations with n items and `num.cycles` cycles.

```
R> count.perms.cycles(perm.length = 6, num.cycles = 4)

[1] 85
```

The following functions are related to the Hamming distance. Recall that an unfixed point is a position of the permutation σ such that $\sigma(i) \neq i$, while a fixed point is a position such that $\sigma(i) = i$. A derangement is a permutation with no fixed point. **PerMallows** includes functions to count the number of permutations of n items with exactly `fixed` fixed points, to count the number of permutations with at least `unfixed` unfixed points and to count the number of permutations with no fixed points.

```
R> count.perms.fixed.points(perm.length = 6, fixed = 4)

[1] 15

R> count.perms.unfixed.points.gtet(perm.length = 6, unfixed = 2)

[1] 504

R> count.derangements(perm.length = 6)

[1] 265
```

The generation of random permutations at a prescribed distance is supported by the `r.dist.d` function, which generates `n` permutations of n items at distance `dist.value` for a particular metric.

```
R> r.dist.d(n = 4, perm.length = 5, dist.value = 3, dist.name = "ulam")

     [,1] [,2] [,3] [,4] [,5]
[1,]    5    3    2    1    4
[2,]    2    5    4    1    3
[3,]    5    4    3    1    2
[4,]    5    3    4    1    2
```

Regarding the Cayley distance, the `r.perms.cycles` function generates uniformly at random permutations of n items with `cycles` cycles.

```
R> r.perms.cycles(n = 3, perm.length = 6, cycles = 5)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    6    5    4
[2,]    4    2    3    1    5    6
[3,]    1    2    4    3    5    6
```

Similarly, `r.derangement` generates `n` permutations with no fixed point.

```
R> r.derangement(n = 5, perm.length = 6)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5    4    1    6    3    2
[2,]    5    1    4    6    2    3
[3,]    4    5    6    2    1    3
[4,]    2    1    6    3    4    5
[5,]    6    3    2    1    4    5
```

3.2. Distributions on permutations

In this section we show how to deal with the Mallows and Generalized Mallows models. We show functions for inference, learning and sampling both models.

Bear in mind that Mallows model (MM) can be used with every metric for permutation considered in this paper, Cayley, Kendall's-τ, Hamming and Ulam. On the other hand, the Generalized Mallows model (GMM) can only be used with Cayley, Kendall's-τ and Hamming[3]. Remember that for distributions on permutations on `perm.length` items the dispersion parameter vector $\boldsymbol{\theta}$ has `perm.length`-1 terms when the distance name is either Cayley or Kendall's-τ, and `perm.length` when the distance is Hamming.

Parameter fitting The estimation of the parameters of the MM and GMM is done with separate functions, `lmm` and `lgmm` respectively. Both need as arguments the sample to fit. The metric to use, an initial guess for the consensus permutation and the estimation methods are optional methods. The latter argument can be `approx` (by default) for the approximate learning and `exact` for the exhaustive one.

```
R> data("data.3")
R> my.mm <- lmm(sample = data.3, sigma_0_ini = identity.permutation(6),
+     dist.name = "cayley", estimation = "exact")
R> my.gmm <- lgmm(sample = data.3, sigma_0_ini = c(2, 1, 6, 5, 3,
+     4), dist.name = "cayley", estimation = "approx")
R> my.mm
```

[3]Although strictly speaking the GMM can not be used with the Hamming distance, we denote the similar Weighted Hamming Mallows model as GMM for the sake of clarity.

```
$mode
[1] 1 2 3 4 5 6

$theta
[1] 0.8364089
```

```
R> my.gmm
```

```
$mode
[1] 1 2 3 4 5 6

$theta
[1] 1.4087672 0.2876821 0.6931472 1.0986123 0.6190392
```

If the auxiliary files have been generated with the function `generate.aux.files`, learning the parameters of the Ulam distance can be done by reading the files instead of computing the necessary data.

```
R> lmm(sample = data.3, dist.name = "ulam", disk = TRUE)
```

```
$mode
[1] 2 5 1 3 4 6

$theta
[1] 0.8900056
```

Generating from the model Although there are two separate functions for sampling MM and GMM (`rmm` and `rgmm`), they are used in a similar way. The required arguments for the MM (GMM) are the number of permutations to be generated, `n`, the parameters of the distribution, that is σ_0 and θ ($\boldsymbol{\theta}$), the name of the distance used and the sampling algorithm (`distances`, `multistage` or `gibbs`).

```
R> rmm(n = 5, sigma0 = my.mm$mode, theta = my.mm$theta, dist.name = "kendall",
+     sampling.method = "distances")
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    4    6    3    5
[2,]    1    2    4    5    3    6
[3,]    2    3    5    1    4    6
[4,]    1    2    3    4    5    6
[5,]    1    3    2    6    4    5
```

```
R> rgmm(n = 5, sigma0 = my.gmm$mode, theta = my.gmm$theta, dist.name = "kendall",
+     sampling.method = "multistage")
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    2    5    4    6
[2,]    1    3    2    5    4    6
[3,]    1    5    2    3    6    4
[4,]    2    3    1    4    5    6
[5,]    2    3    5    1    4    6
```

If the auxiliary files have been generated with the function `generate.aux.files`, the distances sampling for the Ulam distance can rely on them. Again, the `disk` parameter must be set to `TRUE`.

```
R> rmm(n = 3, sigma0 = identity.permutation(6), theta = 1, dist.name = "ulam",
+     disk = TRUE)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    2    4    5    3    6    1
[2,]    1    3    5    2    4    6
[3,]    5    2    3    6    4    1
```

Probability The probability of a permutation for a given MM (GMM) is calculated with the `dmm` (`dgmm`) function. The arguments of `dmm` (`dgmm`) are the permutation, the consensus permutation σ_0, dispersion parameter θ ($\boldsymbol{\theta}$) and the name of the distance used, which by default is Kendall's-τ.

```
R> dmm(perm = my.mm$mode, sigma0 = my.mm$mode, theta = my.mm$theta,
+     dist.name = "ulam")
```

```
[1] 0.01228094
```

```
R> dmm(perm = sigma, sigma0 = my.mm$mode, theta = my.mm$theta, dist.name = "hamming")
```

```
[1] 0.00200106
```

```
R> dgmm(perm = my.gmm$mode, sigma0 = my.gmm$mode, theta = my.gmm$theta,
+     dist.name = "kendall")
```

```
[1] 0.05945101
```

```
R> dgmm(perm = sigma, sigma0 = my.gmm$mode, theta = my.gmm$theta,
+     dist.name = "kendall")
```

```
[1] 0.0003483458
```

In this way, the previous functions can be used to calculate the log-likelihood of a sample of permutations as follows.

```
R> log.prob <- apply(data.3, MARGIN = 1, FUN = function(x) {
+     log(dmm(x, my.mm$mode, my.mm$theta, dist.name = "cayley"))
+ })
R> sum(log.prob)

[1] -123.8265

R> log.prob <- apply(data.3, MARGIN = 1, FUN = function(x) {
+     log(dgmm(x, my.gmm$mode, my.gmm$theta, dist.name = "cayley"))
+ })
R> sum(log.prob)

[1] -122.1056
```

PerMallows includes the computation of the marginal distribution for both MM and GMM under the Hamming distance. The set of fixed and unfixed points is represented in the distance decomposition vector, so $H_j(\sigma) = 0$ means that j is a fixed point, $H_j(\sigma) = 1$ that j is an unfixed point and any other value of $H_j(\sigma)$ means that j is unknown. The following computes the marginal distribution of those permutations having fixed points at positions 2 and 4 and unfixed points at position 1.

```
R> marginal(h = c(1, 0, -1, 0, -1), theta = c(1.1, 2, 1, 0.2, 0.4))

[1] 0.0808545
```

PerMallows includes functions to compute the expectation of the distance (resp. its decomposition vector) under the MM (resp. GMM) under the Hamming distance. The former computes the expectation of the distance under an MM of a given dispersion parameter and number of items. The latter, computes the expectation of the distance decomposition vector given a multidimensional dispersion parameter.

```
R> expectation.mm(theta = 3, perm.length = 9, "ulam")

[1] 1.812466

R> expectation.gmm(c(1.1, 2, 1, 0.2, 0.4), "cayley")

[1] 0.6246747 0.3512144 0.5246331 0.6208475 0.4013123 0.0000000
```

4. Conclusions

This paper describes the associated R package for dealing with probability distributions over permutations spaces, **PerMallows**. The models are the Mallows (MM) and Generalized Mallows (GMM) models. Both models require a distance for permutations. **PerMallows** considers the Kendall's-τ, Cayley, Hamming and Ulam metrics.

The **PerMallows** package is aimed to be a compilation of resources for working on MM and GMM. It provides functions for the exact and approximate estimation of the parameters of a collections of permutations, for simulating from a given distribution or calculating the density function.

Efficient algorithms for reasoning on permutation spaces can not be given with out taking into consideration the particular nature of permutations. Therefore, the core of the **PerMallows** package consists of several functions and operators for permutations such as the factorization of permutations, the random generation of permutations, counting the number of permutations at a given distance, etc.

We expect the **PerMallows** package to be helpful to every kind of user, from the novice in the field of permutations and/or probability models for permutation spaces to the advanced users. Moreover, the internal code has been written in C++ for the maximum efficiency. Also, the code is public for any researcher which wants to extend or improve it.

References

Akl SG (1980). "A New Algorithm for Generating Derangements." *BIT Numerical Mathematics*, **20**(1), 2–7.

Ali A, Meila M (2012). "Experiments with Kemeny Ranking: What works when?" *Mathematical Social Sciences*, **64**(1), 28–40. ISSN 0165-4896.

Arndt J (2010). *Generating Random Permutations*. Ph.D. thesis, Australian National University.

Arora R, Meila M (2013). "Consensus Ranking with Signed Permutations." In *AISTATS*, volume 31 of *JMLR Proceedings*, pp. 117–125. JMLR.org.

Bader M (2011). "The Transposition Median Problem is NP-complete." *Theoretical Computer Science*, **412**(12-14), 1099–1110.

Bartholdi J, Tovey CA, Trick MA (1989). "Voting Schemes for which it Can Be Difficult to Tell who Won the Election." *Social Choice and Welfare*, **6**(2), 157–165.

Borda J (1781). *Memoire Sur les Elections au Scrutin*. Histoire de l'Academie Royal des Sciences.

Ceberio J, Mendiburu A, Lozano JA (2011). "Introducing The Mallows Model on Estimation of Distribution Algorithms." In *International Conference on Neural Information Processing (ICONIP)*, 23-25. Shanghai.

Chen L, Pu P (2004). "Survey of Preference Elicitation Methods."

Cheng W, Hüllermeier E (2009). "A New Instance-Based Label Ranking Approach Using the Mallows Model." In W Yu, H He, N Zhang (eds.), *Advances in Neural Networks - ISNN*, volume 5551 of *Lecture Notes in Computer Science*, pp. 707–716. Springer.

Cheng W, Hullermeier E (2009). "A Simple Instance-Based Approach to Multilabel Classification Using the Mallows Model." In *Workshop Proceedings of Learning from Multi-Label Data*, pp. 28–38. Bled, Slovenia.

Critchlow DE, Fligner MA, Verducci JS (1991). "Probability Models on Rankings." *Journal of Mathematical Psychology*, **35**, 294–318.

D'Elia A, Piccolo D (2005). "A Mixture Model for Preferences Data Analysis." *Computational Statistics & Data Analysis*, **49**(3), 917–934.

Diaconis P (1988). *Group Representations in Probability and Statistics*. Institute of Mathematical Statistics.

Diaconis P (1989). "A Generalization of Spectral Analysis with Application to Ranked Data." *The Annals of Statistics*, **17**(3), 949–979.

Farah M, Vanderpooten D (2007). "An Outranking Approach for Rank Aggregation in Information Retrieval." In *Conference on Research and development in information retrieval (ACM SIGIR)*, SIGIR '07, pp. 591–598. ACM, New York, NY, USA.

Féray V (2013). "Asymptotics of Some Statistics in Ewens Random Permutations." *Electronic Journal of Probability*, **18**(76).

Fligner MA, Verducci JS (1986). "Distance-Based Ranking Models." *Journal of the Royal Statistical Society*, **48**(3), 359–369.

Fligner MA, Verducci JS (1988). "Multistage Ranking Models." *Journal of the American Statistical Association*, **83**(403), 892.

Furnkranz J, Hullermeier E (2013). "Preference Learning and Ranking [Special Issue]." *Machine Learning*, **93**(2-3).

Gnedin A, Olshanski G (2012). "The Two-Sided Infinite Extension of the Mallows Model for Random Permutations." *Advances in Applied Mathematics*, **48**(5), 615–639.

Hatzinger R, Dittrich R (2012). "prefmod: An R Package for Modeling Preferences Based on Paired Comparisons, Rankings, or Ratings." *Journal of Statistical Software*, **48**(10), 1–31.

Irurozki E, Calvo B, Lozano JA (2014a). "Sampling and learning the Mallows and Generalized Mallows models under the Cayley distance." *Technical report*. doi:EHU-KZAA-TR;2014-02. URL https://addi.ehu.es/handle/10810/11239.

Irurozki E, Calvo B, Lozano JA (2014b). "Sampling and learning the Mallows and Weighted Mallows models under the Hamming distance." *Technical report*. doi:EHU-KZAA-TR;2014-03. URL https://addi.ehu.es/handle/10810/11240.

Irurozki E, Calvo B, Lozano JA (2014c). "Sampling and learning the Mallows model under the Ulam distance." *Technical report*. doi:EHU-KZAA-TR;2014-04. URL https://addi.ehu.es/handle/10810/11241.

Lee PH, Yu PLH (2012). "Mixtures of Weighted Distance-Based Models for Ranking Data with Applications in Political Studies." *Computational Statistics & Data Analysis*, **56**(8), 2486–2500.

Lee PH, Yu PLH (2014). ***pmr**: Probability Models for Ranking Data.* URL http://cran.r-project.org/package=pmr.

Mallows CL (1957). "Non-Null Ranking Models." *Biometrika*, **44**(1-2), 114–130.

Mandhani B, Meila M (2009). "Tractable Search for Learning Exponential Models of Rankings." *Journal of Machine Learning Research*, **5**, 392–399.

Mao Y, Lebanon G (2008). "Non-Parametric Modeling of Partially Ranked Data." *Journal of Machine Learning Research*, **9**, 2401–2429.

Meila M, Bao L (2008). "Estimation and Clustering with Infinite Rankings." In *Uncertainty in Artificial Intelligence (UAI)*, pp. 393–402. AUAI Press, Corvallis, Oregon.

Meila M, Chen H (2010). "Dirichlet Process Mixtures of Generalized Mallows Models." In *Uncertainty in Artificial Intelligence (UAI)*, pp. 285–294.

Mueller C, Starr S (2013). "The Length of the Longest Increasing Subsequence of a Random Mallows Permutation." *Journal of Theoretical Probability*, **26**(2), 514–540.

Murphy TB, Martin D (2003). "Mixtures of Distance-Based Models for Ranking Data." *Computational Statistics & Data Analysis*, **41**, 645–655. ISSN 0167-9473.

Popov V (2007). "Multiple Genome Rearrangement by Swaps and by Element Duplications." *Theoretical computer science*, **385**(1-3), 115–126.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992). *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA.

Ziegler A, Christiansen E, Kriegman D, Belongie S (2012). "Locally Uniform Comparison Image Descriptor." In P Bartlett, FCN Pereira, CJC Burges, L Bottou, KQ Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1–9.

Affiliation:

Ekhine Irurozki, Borja Calvo, Jose A. Lozano
E-mail: {ekhine.irurozqui}{borja.calvo}{ja.lozano}@ehu.es
Intelligent Systems Group
Department of Computer Science and Artificial Intelligence
Faculty of Computer Science
University of the Basque Country
20008 Donostia, Spain
URL: http://www.sc.ehu.es/ccwbayes/isg/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd