Journal of Machine Learning Research 12 (2011) 1389-1423 Submitted 8/09; Revised 9/10; Published 4/11

Clustering Algorithms for Chains

Antti Ukkonen AUKKONEN@YAHOO-INC.COM
Yahoo! Research
Av. Diagonal 177
08018 Barcelona, Spain

Editor: Marina Meila

Abstract

We consider the problem of clustering a set of chains to k clusters. A chain is a totally ordered subset of a finite set of items. Chains are an intuitive way to express preferences over a set of alternatives, as well as a useful representation of ratings in situations where the item-specific scores are either difficult to obtain, too noisy due to measurement error, or simply not as relevant as the order that they induce over the items. First we adapt the classical k-means for chains by proposing a suitable distance function and a centroid structure. We also present two different approaches for mapping chains to a vector space. The first one is related to the planted partition model, while the second one has an intuitive geometrical interpretation. Finally we discuss a randomization test for assessing the significance of a clustering. To this end we present an MCMC algorithm for sampling random sets of chains that share certain properties with the original data. The methods are studied in a series of experiments using real and artificial data. Results indicate that the methods produce interesting clusterings, and for certain types of inputs improve upon previous work on clustering algorithms for orders.

Keywords: Lloyd's algorithm, orders, preference statements, planted partition model, randomization testing

1. Introduction

Clustering (see, e.g., Alpaydin, 2004; Hand et al., 2001) is a traditional problem in data analysis. Given a set of objects, the task is to divide the objects to homogeneous groups based on some criteria, typically a distance function between the objects. Cluster analysis has applications in numerous fields, and a myriad of different algorithms for various clustering problems have been developed over the past decades. The reader is referred to the surveys by Xu and Wunsch (2005) and Berkhin (2006) for a more general discussion about clustering algorithms and their applications.

This work is about *clustering a set of orders*, a problem previously studied by Murphy and Martin (2003), Busse et al. (2007), and Kamishima and Akaho (2009). Rankings of items occur naturally in various applications, such as preference surveys, decision analysis, certain voting systems, and even bioinformatics. As an example, consider the Single transferable vote system (Tideman, 1995), where a vote is an ordered subset of the candidates. By clustering such votes, the set of voters can be divided to a number of groups based on their political views. Or, in gene expression analysis it is sometimes of interest to analyze the order of genes induced by the expression levels instead of the actual numeric values (Ben-Dor et al., 2002). In this case a clustering groups genes according to their activity for example under various environmental conditions.

We focus on a particular subclass of (partial) orders, called *chains*. Informally, chains are totally ordered subsets of a set of items, meaning that for all items that belong to a chain we know the order, and for items not belonging to the chain the order is unknown. For example, consider a preference survey about movies where the respondents are requested to rank movies they have seen from best to worst. In this scenario chains are a natural representation for the preference statements, as it is very unlikely that everyone would list the same movies. In a clustering of the responses people with similar preferences should be placed in the same cluster, while people who strongly disagree should be placed in different clusters.

This example also illustrates a very useful property of chains as preference statements: independence of the "scale" used by the respondents when assigning scores to the alternatives. For example, suppose that person A gives movie X three stars, and movie Y five stars. Person B gives movies X and Y one and three stars, respectively. While these ratings are very different, both A and B prefer movie Y to movie X. If we represent a response as a vector of ratings, there is a risk of obtaining clusters that are based on the general level of ratings instead the actual preferences. That is, one cluster might contain respondents who tend to give low ratings, while another cluster contains respondents who give high ratings. Clearly this is not a desirable outcome if the purpose is to study the preferences of the respondents. Statements in the form of chains let us directly focus on the relationships between the alternatives. Moreover, the use of chains can also facilitate preference elicitation, as people may find it easier to rank a small set of items instead of assigning scores to individual items.

Fundamentally the problem of clustering orders does not differ much from the problem of clustering any set of objects for which a distance function can be defined. There are some issues, however. First, *defining a good distance function for chains is not straightforward*. One option is to use existing distance functions for permutations, such as Kendall's tau or Spearman's rho. The usual approach to accommodate these for chains, as taken for example by Kamishima and Akaho (2009), is to only consider the common items of two chains. However, if the chains have no overlap, which can in practice happen quite often, their distance has to be defined in some arbitrary way. The second issue is the *computational complexity* of some of the operations that are commonly used by clustering algorithms. For instance, running Lloyd's algorithm (often called *k*-means) requires the computation of the mean of a set of objects. While this is very easy for numerical inputs and common distance functions, in case of orders one has to solve the rank aggregation problem that is computationally nontrivial; for some choices of the distance function rank aggregation is NP-hard (Dwork et al., 2001). We tackle the aforementioned issues on one hand by formulating the clustering problem in a way that no computationally hard subproblems are involved (Section 2), and on the other hand by by mapping the chains to a vector space (Section 3). By taking the latter approach the problem of clustering chains is reduced to that of clustering vectors in $\mathbb{R}^n$.

In general clustering algorithms will always produce a clustering. However, it is not obvious whether this clustering is reflecting any real phenomena present in the input. Chances are that the output is simply a consequence of random noise. Therefore, in addition to algorithms for finding a clustering, we also propose a method for assessing the validity of the clusterings we find. Our approach falls in the framework of *randomization testing* (Good, 2000), where the statistical significance of a data analysis result is evaluated by running the same analysis on a number of random data sets. If clusterings of a number of random data sets are indistinguishable from a clustering of real data (according to a relevant test statistic), the validity of the clustering found in real data can

be questioned. To make use of this approach we propose a method for generating random sets of chains that share some properties with our original input (Section 4).

1.1 Related Work

Previous research on cluster analysis in general is too numerous to be covered here in full. Instead, we refer the readers to recent surveys by Xu and Wunsch (2005) and Berkhin (2006). For the problem of clustering orders, surprisingly little work has been done. The problem discussed in this paper is also studied by Kamishima and Akaho (2009), and earlier by Kamishima and Fujiki (2003). Murphy and Martin (2003) propose a mixture model for clustering orders. However, they only consider inputs that consist of total orders, that is, every chain in the input must order all items in M. This restriction is not made by Busse et al. (2007) who study a setting similar to ours. An important aspect of their approach is to represent a chain using the set of total orders that are compatible with the chain. This idea can also be found in the work by Critchlow (1985), and is a crucial component of a part of our work in Section 3. Recently Clémençon and Jakubowicz (2010) propose a distance function for permutations based on earth mover's distance between doubly stochastic matrices. While this framework seems quite interesting, extending it for chains seems nontrivial. The use of randomization testing (Good, 2000) in the context of data mining was first proposed by Gionis et al. (2007). Theoretical aspects of the sampling approach are discussed by Besag and Clifford (1989) and Besag and Clifford (1991).

1.2 Organization and Contributions of This Paper

The contributions of this paper are the following:

- In Section 2 we adapt Lloyd's algorithm (Lloyd, 1982) for chains. The main problem is the lack of a good distance function for chains, as well as the computational complexity of rank aggregation. At the core of our approach is to consider the probabilities of pairs of items to precede one another in the cluster.
- In Section 3 we present two methods for mapping chains to high-dimensional vector spaces. The first method aims to preserve the distance between two chains that are assumed to originate from the same component in a simple generative model. The second method represents each chain as the mean of the set of linear extensions of the chain. Our main contribution here is Theorem 5 stating that this can be achieved with a very simple mapping. In particular, it is not necessary to enumerate the set of linear extensions of a chain.
- In Section 4 we present an MCMC algorithm for uniformly sampling sets of chains that share a number of characteristics with a given set of chains. The random sets of chains are used for significance testing.
- In Section 5 we conduct a number of experiments to compare the proposed method with existing algorithms for clustering chains. Turns out that the algorithms are in some sense orthogonal. For smaller data sets the algorithms by Kamishima and Akaho (2009) give in most cases a better result. However, as the input size increases, the method proposed in this paper outperforms other algorithms.

Many of the results presented have appeared previously as a part of the author's doctoral dissertation (Ukkonen, 2008). Theorem 5 in Section 3.2 was presented earlier by Ukkonen (2007) but

Algorithm 1 Lloyd's algorithm

1: k-means(D, k) {Input: D, set of points; k, number of clusters. Output: The clustering $\mathcal{C} = \{D_1, \ldots, D_k\}$.}
2: $\{D_1, \ldots, D_k\} \leftarrow$ PickInitialClusters(D, k);
3: $e \leftarrow \sum_{i=1}^{k} \sum_{x \in D_i} d(\pi, \text{Centroid}(D_i))$;
4: **repeat**
5: $\quad e_0 \leftarrow e$;
6: $\quad \mathcal{C}_0 \leftarrow \{D_1, \ldots, D_k\}$;
7: $\quad$ **for** $i \leftarrow 1, \ldots, k$ **do**
8: $\quad\quad D_i \leftarrow \{x \in D \mid i = \arg\min_j d(x, \text{Centroid}(D_j)\}$;
9: $\quad$ **end for**
10: $\quad e \leftarrow \sum_{i=1}^{k} \sum_{x \in D_i} d(x, \text{Centroid}(D_i))$;
11: **until** $e = e_0$;
12: **return** $\mathcal{C}_0$;

its proof was omitted. Also contents of Section 4 have appeared in less detail in previous work by Ukkonen and Mannila (2007).

2. Adapting Lloyd's Algorithm for Chains

Lloyd's algorithm, also known as k-means, is one of the most common clustering algorithms. In this section we address questions related to the use of Lloyd's algorithm with chains. We start with the basic definitions used throughout this paper.

2.1 Basic Definitions

Let M be a set of m items. A *chain* π is a subset of M together with a total order τ_π on the items, meaning that for every $u, v \in \pi \subseteq M$ we have either $(u,v) \in \tau_\pi$ or $(v,u) \in \tau_\pi$. We use a slightly simplified notation, and say that the pair (u,v) belongs to π, denoted $(u,v) \in \pi$, whenever $(u,v) \in \tau_\pi$. Whenever (u,v) belongs to π, we say that u precedes v according to π. For items in $M \setminus \pi$, the chain π does not specify the order in any way. The chain π is therefore a *partial order*. When π is defined over the entire set M of items, we say it is a *total order*. Let D be a multiset of n chains. A clustering of D is a disjoint partition of D to k subsets, denoted $D_1, \ldots, D_k$, so that every $\pi \in D$ belongs to one and only one D_i.

Lloyd's algorithm (Duda and Hart, 1973; Lloyd, 1982; Ball and Hall, 1967) finds a clustering of $D_1, \ldots, D_k$ so that its *reconstruction error*, defined as

$$\sum_{i=1}^{k} \sum_{x \in D_i} d(x, \text{Centroid}(D_i)), \tag{1}$$

is at a local minimum. Here d is a distance function, D_i is a cluster, and Centroid(D_i) refers to a "center point" of D_i. With numerical data one typically uses the mean as the centroid and squared Euclidean distance as d. The algorithm is given in Algorithm 1. On every iteration Lloyd's algorithm updates the clustering by assigning each point $x \in D$ to the cluster with the closest centroid. The PickInitialClusters function on line 2 of Algorithm 1 can be implemented for example by selecting k total orders at random, and assigning each chain to the the closest one. More sophisticated

techniques, such as the one suggested by Arthur and Vassilvitskii (2007) can also be considered. The algorithm terminates when the clustering error no longer decreases. Note that the resulting clustering is not necessarily a global optima of Equation 1, but the algorithm can end up at a local minimum.

2.2 Problems with Chains

Clustering models are usually based on the concept of distance. In the case of hierarchical clustering we must be able to compute distances between two objects in the input, while with Lloyd's algorithm we have to compute distances to a centroid. Usually the centroid belongs to the same family of objects as the ones in D that we are clustering. However, it can also be something else, and in particular for the problem of clustering chains, *the centroid does not have to be a chain* or even a total order. This is very useful, because defining a good distance function for chains is not straightforward. For example, given the chains $(1,4,5)$ and $(2,3,6)$, it is not easy to say anything about their similarity, as they share no common items. We return to this question later in Section 3.1, but before this we will describe an approach where the distance between two chains is not required.

Another issue arises from the centroid computation. If we use a total order for representing the centroid we have to solve the rank aggregation problem: given all chains belonging to the cluster C_i, we have to compute a total order that is in some sense the "average" of the chains in C_i. This is not trivial, but can be solved by several different approaches. Some of them have theoretical performance guarantees, such as the algorithms by Ailon et al. (2005) and Coppersmith et al. (2006), and some are heuristics that happen to give reasonable results in practice (Kamishima and Akaho, 2006). The hardness of rank aggregation also depends on the distance function. For the Kendall's tau the problem is always NP-hard (Dwork et al., 2001), but for Spearman's rho it can be solved in polynomial time if all chains in the input happen to be total orders. In the general case the problem is NP-hard also for Spearman's rho (Dwork et al., 2001). Our approach is to replace the centroid with a structure that can be computed more efficiently.

2.3 Distances and Centroids

Next we discuss the choice of a centroid and a distance function so that Algorithm 1 can be used directly with an input consisting of chains. Suppose first that the centroid of a cluster is the total order τ. Observe that τ can be represented by a matrix X_τ, where $X_\tau(u,v) = 1$ if and only if we have $(u,v) \in \tau$, otherwise $X_\tau(u,v) = 0$. We can view X_τ as an *order relation*. This relation is completely deterministic, since each pair (u,v) either belongs, or does not belong to τ. Moreover, if (u,v) does not belong to τ, the pair (v,u) has to belong to τ.

A simple generalization of this is to allow the centroid to contain fractional contributions for the pairs. That is, the pair (u,v) may belong to the centroid with a weight that is a value between 0 and 1. We restrict the set of possible weights so that they satisfy the *probability constraint*, defined as $X(u,v) + X(v,u) = 1$ for all $u,v \in M$. In this case the centroid corresponds to a *probabilistic order relation*. Below we show that for a suitable distance function this approach leads to a natural generalization of the case where the centroids are represented by total orders together with Kendall's tau as the distance function. However, this relaxation lets us avoid the rank aggregation problem discussed above.

Consider the following general definition of a centroid. Given a set D of objects and the class Q of centroids for D, we want to find a $X^* \in Q$, so that

$$X^* = \arg\min_{X \in Q} \sum_{\pi \in D} d(\pi, X),$$

where $d(\pi, X)$ is a distance between π and X. Intuitively X^* must thus reside at the "center" of the set D. We let Q be set of probabilistic order relations on M, that is, the set of $|M| \times |M|$ matrices satisfying the probability constraint. Given a matrix $X \in Q$ and a chain π, we define the distance $d(\pi, X)$ as

$$d(\pi, X) = \sum_{(u,v) \in \pi} X(v,u)^2. \tag{2}$$

This choice of $d(\pi, X)$ leads to a simple way of computing the optimal centroid, as is shown below. Note that this distance function is equivalent with Kendall's tau if X is a deterministic order relation. To find the centroid of a given set D of chains, we must find a matrix $X \in Q$ such that the cost

$$c(X, D) = \sum_{\pi \in D} \sum_{(u,v) \in \pi} X(v,u)^2$$

is minimized. By writing the sum in terms of pairs of items instead of chains, we obtain

$$c(X, D) = \sum_{u \in M} \sum_{v \in M} C_D(u,v) X(v,u)^2,$$

where $C_D(u,v)$ denotes the number of chains in D where u appears before v. Let U denote the set of all unordered pairs of items from M. Using U the above can be written as

$$c(X, D) = \sum_{\{u,v\} \in U} \left(C_D(u,v) X(v,u)^2 + C_D(v,u) X(u,v)^2 \right).$$

As X must satisfy the probability constraint, this becomes

$$c(X, D) = \sum_{\{u,v\} \in U} \underbrace{\left(C_D(u,v)(1 - X(u,v))^2 + C_D(v,u) X(u,v)^2 \right)}_{c(X, \{u,v\})}. \tag{3}$$

To minimize Equation 3 it is enough to independently minimize the individual parts of the sum corresponding to the pairs in U, denoted $c(X, \{u,v\})$. Setting the first derivative of this with respect to $X(u,v)$ equal to zero gives

$$X^*(u,v) = \frac{C_D(u,v)}{C_D(u,v) + C_D(v,u)}. \tag{4}$$

That is, the optimal centroid is represented by a matrix X^* where $X^*(u,v)$ can be seen as a simple estimate of the probability of item $u \in M$ to precede item $v \in M$ in the input D. This is a natural way of expressing the the ordering information present in a set of chains without having to construct an explicit total order.

It is also worth noting that long chains will be consistently further away from the centroid than short chains, because we do not normalize Equation 2 with the length of the chain. This is not a problem, however, since we are only using the distance to assign a chain to one of the k centroids.

Distances of two chains of possibly different lengths are not compared. We also emphasize that even if longer chains in some sense contribute more to the centroid, as they contain a larger number of pairs, the contribution to an individual element of the matrix X is independent of chain length.

We propose thus to use Lloyd's algorithm as shown in Algorithm 1 with the distance function in Equation 2 and the centroid as defined by Equation 4. The algorithm converges to a local optimum, as the reconstruction error decreases on every step. When assigning chains to updated centroids the error can only decrease (or stay the same) because the chains are assigned to clusters that minimize the error (line 8 of Alg. 1). When we recompute the centroids given the new assignment of chains to clusters, the error is non-increasing as well, because the centroid X^* (Equation 4) by definition minimizes the error for every cluster.

3. Mappings to Vector Spaces

In this section we describe an alternative approach to clustering chains. Instead of operating directly on the chains, we first map them to a vector space. This makes it possible to compute the clustering using *any* algorithm that clusters vectors. Note that this will lead to a clustering that does not minimize the same objective function as the algorithm described in the previous section. However, the two approaches are complementary: we can first use the vector space representation to compute an initial clustering of the chains, and then refine this with Lloyd's algorithm using the centroid and distance function of the previous section. Note that these mappings can also be used to visualize sets of chains (Ukkonen, 2007; Kidwell et al., 2008).

3.1 Graph Representation

The mapping that we describe in this section is based on the adjacency matrices of two graphs where the chains of the input D appear as vertices. These graphs can be seen as special cases of the so called planted partition model (Condon and Karp, 2001; Shamir and Tsur, 2002).

3.1.1 MOTIVATION

We return to the question of computing the distance between two chains. Both Spearman's rho and Kendall's tau can be modified for chains so that they only consider the common items. If the chains π_1 and π_2 have no items in common, we have to use a fixed distance between π_1 and π_2. This is done for example by Kamishima and Fujiki (2003), where the distance between two chains is given by $1-\rho$, where $\rho \in [-1,1]$ is Spearman's rho. For two fully correlated chains the distance becomes 0, and for chains with strong negative correlation the distance is 2. If the chains have no common items we have $\rho = 0$ and the distance is 1. We could use the same approach also with the Kendall distance by defining the distance between the chains π_1 and π_2 as the (normalized) Kendall distance between the permutations that are induced by the common items in π_1 and π_2. If there are no common items we set the distance to 0.5. Now consider the following example. Let $\pi_1 = (1,2,3,4,5)$, $\pi_2 = (6,7,8,9,10)$, and $\pi_3 = (4,8,2,5,3)$. By definition we have $d_K(\pi_1,\pi_2) = 0.5$, and a simple calculation gives $d_K(\pi_1,\pi_3) = 0.5$ as well. Without any additional information this is a valid approach.

However, suppose that the input D has been generated by the following model: We are given k partial orders Π_j, $j = 1,\ldots,k$, on M. A chain π is generated by first selecting one of the Π_js at random, then choosing one linear extension τ of Π_j at random, and finally picking a random subset

of l items and creating the chain by projecting τ on this subset. (This model is later used in the experiments in Section 5).

Continuing the example, let π_1, π_2, and π_3 be defined as above, assume for simplicity that the Π_js of the generative model are total orders, and that π_1 and π_2 have been generated by the same component, the total order $(1,2,3,4,5,6,7,8,9,10)$, and that π_3 is generated by another component, the total order $(6,7,9,10,4,8,2,5,3,1)$. Under this assumption it no longer appears meaningful to have $d_K(\pi_1,\pi_2) = d_K(\pi_1,\pi_3)$, as the clustering algorithm should separate chains generated by different components from each other. We would like to have $d_K(\pi_1,\pi_2) < d_K(\pi_1,\pi_3)$. Of course we can a priori not know the underlying components, but when computing a clustering we are assuming that they exist.

3.1.2 AGREEMENT AND DISAGREEMENT GRAPHS

Next we propose a method for mapping the chains to $\mathbb{R}^n$ so that the distances between the vectors that correspond to π_1, π_2 and π_3 satisfy the inequality of the example above. In general we want chains that are generated by the same component to have a shorter distance to each other than to chains that originate from other components. To this end, we define the distance between two chains in D as the distance between their neighborhoods in appropriately constructed graphs. If the neighborhoods are similar, that is, there are many chains in D that are (in a sense to be formalized shortly) "close to" both π_1 and π_2, we consider also π_1 and π_2 similar to each other. Note that this definition of distance between two chains is dependent on the input D. In other words, the distance between π_1 and π_2 can change if other chains in D are modified.

We say that chains π_1 and π_2 *agree* if for some items u and v we have $(u,v) \in \pi_1$ and $(u,v) \in \pi_2$. Likewise, the chains π_1 and π_2 *disagree* if for some u and v we have $(u,v) \in \pi_1$ and $(v,u) \in \pi_2$. Note that π_1 and π_2 can simultaneously both agree and disagree. We define the agreement and disagreement graphs:

Definition 1 *Let $G_a(D)$ and $G_d(D)$ be undirected graphs with chains in D as vertices. The graph $G_a(D)$ is the* agreement graph, *where two vertices are connected by an edge if their respective chains* agree and do not disagree. *The graph $G_d(D)$ is the* disagreement graph, *where two vertices are connected by an edge if their respective chains* disagree and do not agree.

The distance between chains π_1 and π_2 will be a function of the sets of neighboring vertices of π_1 and π_2 in $G_a(D)$ and $G_d(D)$. Before giving the precise definition we discuss some theory related to the graph $G_a(D)$. This will shed some light on the hardness of finding a clustering if the input D is very sparse.

3.1.3 THE PLANTED PARTITION MODEL

Consider the following stochastic model for creating a random graph of n vertices. First partition the set of vertices to k disjoint subsets denoted $V_1,\ldots,V_k$. Then, independently generate edges between the vertices as follows: add an edge between two vertices that belong to the same subset with probability p, and add an edge between two vertices that belong to different subsets with probability $q < p$. This model, called the planted partition model, was first discussed by Condon and Karp (2001) and subsequently by Shamir and Tsur (2002). They also proposed algorithms for recovering the underlying clustering as long as the gap $\Delta = p - q$ is not too small.

Assuming a simple process that generates the input D we can view the agreement graph $G_a(D)$ as an instance of the planted partition model with values of p and q that depend on the characteristics

of the input D. More specifically, let D be generated by k total orders on the set of items M, so that each chain $\pi \in D$ is the projection of one of the total orders on some l-sized subset of M. In theory we can compute a clustering of D by applying one of the existing algorithms for the planted partition model on the graph $G_a(D)$. However, this approach may fail in practice. We argue that for realistic inputs D the graph $G_a(D)$ is unlikely to satisfy the condition on the gap Δ required by the algorithms given by Condon and Karp (2001) and Shamir and Tsur (2002). Also, these algorithms are rather complex to implement.

We start by considering the probability of observing an edge between two vertices in the graph $G_a(D)$ when D is generated using the model outlined above. This happens when two independent events are realized. First, the chains corresponding to the vertices must have at least 2 common items, the probability of which we denote by $\Pr(|\pi_1 \cap \pi_2| \geq 2)$. Observe that this is the disjoint union of events where there are exactly i common items, $i \in [2, l]$. Therefore, we have $\Pr(|\pi_1 \cap \pi_2| \geq 2) = \sum_{i=2}^{l} \Pr(|\pi_1 \cap \pi_2| = i)$. Second, the common items must be ordered in the same way in both of the chains. Denote the probability of this by $\Pr(\pi_1 \perp_i \pi_2)$ for the case of i common items. The probability of observing an edge between π_1 and π_2 is thus given by the sum

$$\sum_{i=2}^{l} \Pr(|\pi_1 \cap \pi_2| = i) \Pr(\pi_1 \perp_i \pi_2). \tag{5}$$

Next we use this to derive the probabilities p and q of observing an edge between two chains that belong either to the same, or two different components, respectively. Clearly we have $\Pr(|\pi_1 \cap \pi_2| = i) = \binom{l}{i}\binom{m-l}{l-i}\binom{m}{l}^{-1}$ in both cases, as the number of common items is independent of their ordering. The only part that matters is thus $\Pr(\pi_1 \perp_i \pi_2)$. When π_1 and π_2 belong to the *same component*, this probability is equal to 1, because π_1 and π_2 are always guaranteed to order every subset of items in the same way. Hence Equation 5 gives

$$p = \binom{m}{l}^{-1} \sum_{i=2}^{l} \binom{l}{i}\binom{m-l}{l-i}. \tag{6}$$

When π_1 and π_2 belong to *different components*, we must make sure that the component that emits π_2 orders the common items in the same way as π_1. (To simplify matters we allow the second component to be identical to the one that has generated π_1. This will not significantly affect the subsequent analysis.) The number of permutations on m items where the order of i items is fixed is $m!/i!$. Since the component of π_2 is sampled uniformly at random from all possible permutations, we have $\Pr(\pi_1 \perp_i \pi_2) = \frac{m!}{i!m!} = 1/i!$. This together with Equation 5 yields

$$q = \binom{m}{l}^{-1} \sum_{i=2}^{l} \frac{\binom{l}{i}\binom{m-l}{l-i}}{i!}. \tag{7}$$

The algorithm of Condon and Karp (2001) requires a gap Δ of order $\Omega(n^{-\frac{1}{2}+\varepsilon})$ given an input of size n to find the correct partitioning (for $k = 2$). The improved algorithm by Shamir and Tsur (2002) is shown to produce a correct output with Δ of order $\Omega(kn^{-\frac{1}{2}}\log n)$. Another way of seeing these results is that as Δ decreases more and more data is needed (n must increase) for the algorithms to give good results. Next we study how the gap Δ behaves in $G_a(D)$ as a function of $m = |M|$ and the length l of the chains. (Assuming that all chains are of equal length.) Since we have

$$\Delta = p - q = \frac{\sum_{i=2}^{l} \binom{l}{i}\binom{m-l}{l-i}\left(1 - \frac{1}{i!}\right)}{\binom{m}{l}},$$

where $(1-\frac{1}{i!})$ is significantly less than 1 only for very small i (say, $i \leq 3$), it is reasonable to bound Δ by using an upper bound for p. We obtain the following theorem:

Theorem 2 *Let p and q be defined as in Equations 6 and 7, respectively, and let $\Delta = p - q$. For $l < m/2$, we have*

$$\Delta < p = O\Big(\frac{l^2}{m}\Big).$$

Proof See Appendix A.1. ■

The bound expresses how the density of the graph $G_a(D)$ depends on the number of items m and the length of the chains l. The gap Δ becomes smaller as m increases and l decreases. This, combined with the existing results concerning Δ, means that for short chains over a large M the input D has to be very large for the algorithms of Condon and Karp (2001) and Shamir and Tsur (2002) to produce good results. For example with $l = 5$ and $m = 200$, Theorem 2 gives an upper bound of $1/8$ for Δ. But for example the algorithm of Shamir and Tsur (2002) requires Δ to be lower bounded by $kn^{-\frac{1}{2}}\log(n)$ (up to a constant factor). To reach $1/8$ with $k = 2$, n must in this case be of order 10^5, which can be tricky for applications such as preference surveys. Therefore, we conclude that for these algorithms to be of practical use a relatively large number of chains is needed if the data consists of short chains over a large number of different items. Also, even though Theorem 2 is related to the graph $G_a(D)$, it gives some theoretical justification to the intuition that increasing the length of the chains should make the clusters easier to separate.

3.1.4 USING $G_a(D)$ AND $G_d(D)$

In the agreement graph, under ideal circumstances the chain π is mostly connected to chains generated by the same component as π. Also, it is easy to see that in the disagreement graph the chain π is (again under ideal circumstances) not connected to *any* of the chains generated by the same component, and only to chains generated by the other components. This latter fact makes it possible to find the correct clustering by finding a k-coloring of $G_d(D)$. Unfortunately this has little practical value as in real data sets we expect to observe noise that will distort both $G_a(D)$ and $G_d(D)$.

Above we argued that representations of two chains emitted by the same component should be more alike than representations of two chains emitted by different components. Consider the case where $k = 2$ and both clusters are of size $n/2$. Let $f_\pi \in \mathbb{R}^n$ be the row of the adjacency matrix of $G_a(D)$ that corresponds to chain π. Let chain π_1 be generated by the same component as π, and let π_2 be generated by a different component. Also, define the similarity s between f_π and $f_{\pi'}$ as the number of elements where both f_π and $f_{\pi'}$ have the value 1. Consider the expected value of this similarity under the planted partition model. We have:

$$\begin{aligned} E[s(f_\pi, f_{\pi_1})] &= \frac{n}{2}p^2 + \frac{n}{2}q^2 = \frac{n}{2}(p^2+q^2), \\ E[s(f_\pi, f_{\pi_2})] &= \frac{n}{2}pq + \frac{n}{2}qp = nqp. \end{aligned}$$

It is easy to see that $E[s(f_\pi, f_{\pi_1})] > E[f_\pi, f_{\pi_2}]$ if we let $p = cq$, with $c > 1$. (This is true if p and q are defined as in Equations 6 and 7.) Therefore, at least under these simple assumptions the expected distance between two chains from the same component is always less than the expected distance between two chains from different components. In practice we can combine the adjacency matrices of $G_a(D)$ and $G_d(D)$ to create the final mapping:

Definition 3 *Let $G_{ad} = G_a(D) - G_d(D)$, where $G_a(D)$ and $G_d(D)$ denote the adjacency matrices of the agreement and disagreement graphs. The representation of the chain π in $\mathbb{R}^n$ is the row of G_{ad} that corresponds to π.*

While the analysis above only concerns $G_a(D)$, we chose to combine both graphs in the final representation. This can be motivated by the following example. As above, let f_π denote the row of the adjacency matrix of $G_a(D)$ that corresponds to the chain π, and let g_π denote the same for $G_d(D)$. Suppose that the chain π_1 agrees with the chain π, meaning that $f_{\pi_1}(\pi) = 1$ and $g_{\pi_1}(\pi) = 0$, and let the chain π_2 disagree with π, meaning that $f_{\pi_2}(\pi) = 0$ and $g_{\pi_2}(\pi) = 1$. Also, assume that the chain π_3 neither agrees nor disagrees with π, meaning that $f_{\pi_3}(\pi) = g_{\pi_3}(\pi) = 0$. Intuitively, in this example the distance between π_1 and π_2 should be larger than the distance between π_1 and π_3. With $G_{ad}(D)$ this property is satisfied, as now in the final representations, defined as $h_{\pi_i} = f_{\pi_i} - g_{\pi_i}$, we have $h_{\pi_1}(\pi) = 1$, $h_{\pi_2}(\pi) = -1$, and $h_{\pi_3}(\pi) = 0$. Using only $G_a(D)$ fails to make this distinction, because $f_{\pi_2}(\pi) = f_{\pi_3}(\pi)$.

Using the agreement and disagreement graphs has the obvious drawback that the adjacency matrices of $G_a(D)$ and $G_d(D)$ are both of size $n \times n$, and computing one entry takes time proportional to l^2. Even though $G_a(D)$ and $G_d(D)$ have the theoretically nice property of being generated by the planted partition model, using them in practice can be prohibited by these scalability issues. However, there is some experimental evidence that the entire G_{ad} graph is not necessarily needed (Ukkonen, 2008).

3.2 Hypersphere Representation

Next we devise a method for mapping chains to an m-dimensional (as opposed to n-dimensional) vector space. The mapping can be computed in time $O(nm)$. This method has a slightly different motivation than the one discussed above. Let f be the mapping from the set of all chains to $\mathbb{R}^m$ and let d be a distance function in $\mathbb{R}^m$. Furthermore, let π be a chain and denote by π^R the reverse of π, that is, the chain that orders the same items as π, but in exactly the opposite way. The mapping f and distance d should satisfy

$$d(f(\pi), f(\pi^R)) = \max_{\pi'} \{d(f(\pi), f(\pi'))\} \quad (8)$$

$$d(f(\pi_1), f(\pi_1^R)) = d(f(\pi_2), f(\pi_2^R)) \text{ for all } \pi_1 \text{ and } \pi_2. \quad (9)$$

Less formally, we want the reversal of a chain to be furthest away from it in the vector space (8), and the distance between π and π^R should be the same for all chains (9). We proceed by first defining a mapping for total orders that satisfy the conditions above and then generalize this for chains. In both cases the mappings have an intuitive geometrical interpretation.

3.2.1 A MAPPING FOR TOTAL ORDERS

We define a function f that maps total orders to $\mathbb{R}^m$ as follows: Let τ be a total order on M, and let $\tau(u)$ denote the position of $u \in M$ in τ. For example, if $M = \{1, \ldots, 8\}$ and $\tau = (5,1,6,3,7,2,8,4)$, we have $\tau(5) = 1$. Consider the vector $\mathbf{f}_\tau$ where

$$\mathbf{f}_\tau(u) = -\frac{m+1}{2} + \tau(u) \quad (10)$$

for all $u \in M$. We define the mapping f such that $f(\tau) = \mathbf{f}_\tau / \|\mathbf{f}_\tau\| = \hat{\mathbf{f}}_\tau$. Note that this mapping is a simple transformation of the Borda count (see, e.g., Moulin, 1991), where candidates in an election

are given points based on their position in the order specified by a vote. Returning to the example, according to Equation 10 we have

$$\mathbf{f}_\tau = (-2.5, 1.5, -0.5, 3.5, -3.5, -1.5, 0.5, 2.5),$$

and as $\|\mathbf{f}_\tau\| = 6.48$, we have

$$f(\tau) = \hat{\mathbf{f}}_\tau = (-0.39, 0.23, -0.08, 0.54, -0.54, -0.23, 0.08, 0.39).$$

When d is the *cosine distance* between two vectors, which in this case is simply $1 - \hat{\mathbf{f}}_\tau^T \hat{\mathbf{f}}_{\tau'}$ as the vectors are normalized, it is straightforward to check that $\hat{\mathbf{f}}_\tau$ satisfies Equations 8 and 9. This mapping has a geometrical interpretation: all permutations are points on the surface of an m-dimensional unit-sphere centered at the origin. Moreover, the permutation τ and its reversal τ^R are on exactly opposite sides of the sphere. That is, the image of τ^R is found by mirroring the image of τ at the origin.

3.2.2 A MAPPING FOR CHAINS

To extend the above for chains we apply the technique used also by Critchlow (1985) and later by Busse et al. (2007). The idea is to represent a chain π on M by the set of total orders on M that are compatible with π. That is, we view π as a partial order on M and use the set of linear extensions[1] of π to construct the representation $f(\pi)$. More precisely, we want $f(\pi)$ to be the *center* of the points in the set $\{f(\tau) : \tau \in \mathcal{E}(\pi)\}$, where f is the mapping for permutations defined in the previous section, and $\mathcal{E}(\pi)$ is the set of linear extensions of π. Our main contribution in this section is that despite the size of $\mathcal{E}(\pi)$ is $\binom{m}{l}(m-l)!$, we can compute $f(\pi)$ very efficiently. We start by giving a definition for $f(\pi)$ that is unrelated to $\mathcal{E}(\pi)$.

Definition 4 *Let π be a chain over M and define the vector $\mathbf{f}_\pi$ so that*

$$\mathbf{f}_\pi(u) = \begin{cases} -\frac{|\pi|+1}{2} + \pi(u) & \text{iff } u \in \pi, \\ 0 & \text{iff } u \notin \pi, \end{cases} \tag{11}$$

for all $u \in M$. The mapping f is defined so that $f(\pi) = \mathbf{f}_\pi / \|\mathbf{f}_\pi\| = \hat{\mathbf{f}}_\pi$.

This is a generalization of the mapping for total orders to the case where only a subset of the items has been ordered. The following theorem states that this definition makes $f(\pi)$ the center of the set $\{f(\tau) : \tau \in \mathcal{E}(\pi)\}$.

Theorem 5 *If the vector $\mathbf{f}_\tau$ is defined as in Equation 10, and the vector $\mathbf{f}_\pi$ is defined as in Equation 11, then there exists a constant Q so that*

$$\mathbf{f}_\pi(u) = Q \sum_{\tau \in \mathcal{E}(\pi)} \mathbf{f}_\tau(u) \tag{12}$$

for all $u \in M$.

1. A linear extension of a partial order π is a total order τ so that $(u,v) \in \pi \rightarrow (u,v) \in \tau$.

Proof See Appendix A.2. ■

What does this theorem mean in practice? We want $f(\pi)$ to be the mean of the points that represent the linear extensions of π, normalized to unit length. Theorem 5 states that this mean has a simple explicit formula that is given by Equation 11. Thus, when normalizing $\mathbf{f}_\pi$ we indeed get the normalized mean vector *without having to sum over all linear extensions of* π. This is very important, as $\mathcal{E}(\pi)$ is so large that simply enumerating all its members is computationally infeasible.

The first advantage of the hypersphere representation over the agreement and disagreement graphs is efficiency. Computing the vectors $\mathbf{f}_\pi$ for all chains in the input is of order $O(nm)$, which is considerably less than the requirement of $O(n^2m^2)$ for the graph based approach. As a downside we lose the property of having a shorter distance between chains generated by the same component than between chains generated by different components. The second advantage of the hypersphere mapping is size. Storing the full graph representation requires $O(n^2)$ memory, while storing the hypersphere representation needs only $O(nm)$ of storage. This is the same as needed for storing D, and in most cases less than $O(n^2)$ as usually we have $m \ll n$.

4. Assessing the Significance of Clusterings

Clustering algorithms will in general always produce a clustering of the input objects. However, it is not obvious that these clusterings are meaningful. If we run one of the algorithms discussed above on a random set of chains, we obtain a clustering as a result. But clearly this clustering has in practice no meaning. To assess the significance of a clustering of the input D, we compare its reconstruction error with the errors of clusterings obtained from random (in a sense made precise below) sets of chains. If the error from real data is smaller than the errors from random data, we have evidence for the clustering to be meaningful. The random sets of chains must share certain aspects with our original input D. In this section we define these aspects precisely, and devise a method for sampling randomized sets of chains that share these aspects with a given input D.

4.1 Randomization Testing and Empirical p-values

For a thorough discussion of randomization testing, we refer the reader to the textbook by Good (2000). Below we give only a brief outline and necessary definitions. Denote by $\mathcal{A}$ a data analysis algorithm that takes D as the input and produces some output, denoted $\mathcal{A}(D)$. We can assume that $\mathcal{A}(D)$ is in fact the value of a *test statistic* that we are interested in. For the remainder of this paper $\mathcal{A}$ is a clustering algorithm and $\mathcal{A}(D)$ is the reconstruction error of the clustering found by $\mathcal{A}$. Moreover, denote by $\tilde{D}_1, \ldots, \tilde{D}_h$ a sequence of random sets of chains that share certain properties with D. These will be defined more formally later.

If the value $\mathcal{A}(D)$ considerably deviates from the values $\mathcal{A}(\tilde{D}_1), \ldots, \mathcal{A}(\tilde{D}_h)$, we have some evidence for the output of $\mathcal{A}$ to be meaningful. In practice this means we can rule out the common properties of the real and random data sets as the sole causes for the results found. As usual in statistical testing we can speak of a *null hypothesis* H_0 and an *alternative hypothesis* H_1. These are defined as follows:

$$\begin{aligned} H_0 : \mathcal{A}(D) &\geq \min_i \{\mathcal{A}(\tilde{D}_i)\}, \\ H_1 : \mathcal{A}(D) &< \min_i \{\mathcal{A}(\tilde{D}_i)\}. \end{aligned}$$

In statistics the *p-value* of a test usually refers to the probability of making an error when rejecting H_0 (and accepting H_1). In order to determine the p-value one typically needs to make some assumptions of the distribution of the test statistic. In general, if we cannot, or do not want to make such assumptions, we can compute the *empirical p-value* based on the randomized data sets. This is defined simply as the fraction of cases where the value of $\mathcal{A}(\tilde{D}_i)$ is more extreme than the value $\mathcal{A}(D)$. Or more formally, for the one-tailed case where $\mathcal{A}(D)$ is expected to be small according to H_1, we have

$$\hat{p} = \frac{|\{\tilde{D}_i : \mathcal{A}(\tilde{D}_i) \leq \mathcal{A}(D)\}| + 1}{h+1}.$$

One problem with using $\hat{p}$ is that in order to get useful values the number of randomized data sets must be fairly high. For instance, to have $\hat{p} = 0.001$ we must sample at least 999 data sets. Depending on the complexity of generating one random data set this may be difficult. Of course, already with 99 data sets we can obtain an empirical p-value of 0.01 if all random data sets have a larger value of the test statistic. This should be enough for many practical applications.

4.2 Equivalence Classes of Sets of Chains

The random data sets must share some characteristics with the original data D. Given D, we define an equivalence class of sets of chains, so that all sets belonging to this equivalence class have the same properties as D.

Definition 6 *Let D_1 and D_2 be two sets of chains on items of the set M. D_1 and D_2 belong to the same equivalence class whenever the following three conditions hold.*

1. *The number of chains of length l is the same in D_1 as in D_2 for all l.*
2. *For all $M' \subseteq M$, the number of chains that contain M' as a subset is the same in D_1 and D_2.*
3. *We have $C_{D_1}(u,v) = C_{D_2}(u,v)$ for all $u,v \in M$, where $C_D(u,v)$ is the number of chains in D that rank u before v.*

Given a set D of chains, we denote the equivalence class specified by D with $\mathcal{C}(D)$. Next we discuss an algorithm for sampling uniformly from $\mathcal{C}(D)$. But first we elaborate why it is useful to maintain the properties listed above when testing the significance of $\mathcal{A}(D)$.

When analyzing chains over the items in M, the most interesting property is how the chains actually order the items. In other words, the clustering should reflect the *ordering information* present in D. This is only one property of D, however. Other properties are those that we mention in the conditions above. Condition 1 is used to rule out the possibility that the value of $\mathcal{A}(D)$ is somehow caused only by the length distribution of the chains in D. Note that this requirement also implies that D_1 and D_2 are of the same size. Likewise, condition 2 should rule out the possibility that the result is not a consequence of the rankings, but simply the co-occurrences of the items.

Maintaining $C_D(u,v)$ is motivated from a slightly different point of view. If D contained real-valued vectors instead of chains, it would make sense to maintain the empirical mean of the observations. The intuition with chains is the same: we view D as a set of points in the space of chains. The random data sets should be located in the same region of this space as D. By maintaining $C_D(u,v)$ the randomized data sets $\tilde{D}_i$ will (in a way) have the same mean as D. This is because the rank aggregation problem, that is, finding the mean of a set of permutations, can be solved using only the $C_D(u,v)$ values (Ukkonen, 2008).

4.3 An MCMC Algorithm for Sampling from $\mathcal{C}(D)$

Next we will discuss a Markov chain Monte Carlo algorithm that samples uniformly from a subset of $\mathcal{C}(D)$ given D. We can only guarantee that the sample will be from a neighborhood of D in $\mathcal{C}(D)$. Whether this neighborhood covers all of $\mathcal{C}(D)$ is an open problem.

4.3.1 ALGORITHM OVERVIEW

The MCMC algorithm we propose can be seen as a random walk on an undirected graph with $\mathcal{C}(D)$ as the set of vertices. Denote this graph by $G(D)$. The vertices D_1 and D_2 of $G(D)$ are connected by an edge if we obtain D_2 from D_1 by performing a small local modification to D_1 (and vice versa). We call this local modification a *swap* and will define it in detail below. First, let us look at a high level description of the algorithm.

In general, when using MCMC to sample from a distribution, we must construct the Markov Chain so that its *stationary distribution* equals the target distribution we want to sample from. If all vertices of $G(D)$ are of equal degree, the stationary distribution will be the uniform distribution. As we want to sample uniformly from $\mathcal{C}(D)$, this would be optimal. However, it turns out that the way we have defined the graph $G(D)$ does not result in the vertices having the same number of neighboring vertices. To remedy this, we use the *Metropolis-Hastings* algorithm (see, e.g., Gelman et al., 2004) for picking the next state. Denote by $N(D_i)$ the set of neighbors of the vertex D_i in $G(D)$. When the chain is at D_i, we pick uniformly at random the vertex D_{i+1} from $N(D_i)$. The chain moves to D_{i+1} with probability

$$\min\left(\frac{|N(D_i)|}{|N(D_{i+1})|}, 1\right), \tag{13}$$

that is, the move is accepted always when D_{i+1} has a smaller degree, and otherwise we move with a probability that decreases as the degree of D_{i+1} increases. If the chain does not move, it stays at the state D_i and attempts to move again (possibly to some other neighboring vertex) in the next step.

It is easy to show that this modified random walk has the desired property of converging to a uniform distribution over the set of vertices. Denote by $p(D_i)$ the *target distribution* we want to sample from. In this case $p(D_i)$ is the uniform distribution over $\mathcal{C}(D)$. Hence, we must have $p(D_i) = p(D_{i+1}) = |\mathcal{C}(D)|^{-1}$. The Metropolis-Hastings algorithm jumps to the next state D_{i+1} with probability $\min(r, 1)$, where

$$r = \frac{p(D_{i+1})/J(D_{i+1}|D_i)}{p(D_i)/J(D_i|D_{i+1})}. \tag{14}$$

Above $J(\cdot|\cdot)$ is a *proposal distribution*, which in this case is simply the uniform distribution over the neighbors of D_i for all i. That is, we have $J(D_{i+1}|D_i) = |N(D_i)|^{-1}$ and $J(D_i|D_{i+1}) = |N(D_{i+1})|^{-1}$. When this is substituted into Equation 14 along with the fact that $p(D_i) = p(D_{i+1})$ we obtain Equation 13.

Given D, a simple procedure for sampling one $\tilde{D}$ uniformly from $\mathcal{C}(D)$ works as follows: we start from $D = D_0$, run the Markov chain resulting in slightly modified data D_i on every step i. After s steps we are at the set D_s which is our $\tilde{D}$. We repeat this process until enough samples from $\mathcal{C}(D)$ have been obtained. It is very important to run the Markov chain long enough (have a large enough s), so that the samples are as uncorrelated as possible with the starting point D, as well as independent of each other. We will discuss a heuristic for assessing the correct number steps below.

However, guaranteeing that the samples are independent is nontrivial. Therefore we only require the samples to be *exchangeable*. The following approach, originally proposed by Besag and Clifford (1989), draws h sets of chains from $\mathcal{C}(D)$ so that the samples satisfy the exchangeability condition. We first start the Markov chain from D and run it *backwards* for s steps. (In practice the way we define our Markov chain, running it backwards is equivalent to running it forwards.) This gives us the set $\tilde{D}_0$. Next, we run the chain forwards $h-1$ times for s steps, each time starting from $\tilde{D}_0$. This way the samples are not dependent on each other, but only on $\tilde{D}_0$. And since we obtained $\tilde{D}_0$ by running the Markov chain backwards from D, the samples depend on $\tilde{D}_0$ in the same way as D depends on $\tilde{D}_0$. Note that a somewhat more efficient approach is proposed by Besag and Clifford (1991).

4.3.2 THE SWAP

Above we defined the Markov chain as a random walk over the elements of $\mathcal{C}(D)$, where two states D and D' are connected if one can be obtained from the other by a local modification operator. We call this local modification a *swap* for reasons that will become apparent shortly. Since the Markov chain must remain in $\mathcal{C}(D)$, the swap may never result in a set of chains $\hat{D} \notin \mathcal{C}(D)$. More precisely, if D_{i+1} is obtained from D_i by the swap and $D_i \in \mathcal{C}(D)$, then D_{i+1} must belong to $\mathcal{C}(D)$ as well. Next we define a swap that has this property.

Formally we define a swap as the tuple (π_1, π_2, i, j), where π_1 and π_2 are chains, i is an index of π_1, and j an index of π_2. To execute the swap (π_1, π_2, i, j), we transpose the items at positions i and $i+1$ in π_1, and at positions j and $j+1$ in π_2. For example, if $\pi_1 = (1,2,3,4,5)$ and $\pi_2 = (3,2,6,4,1)$, the swap $(\pi_1, \pi_2, 2, 1)$ will result in the chains $\pi_1' = (1,3,2,4,5)$ and $\pi_2' = (2,3,6,4,1)$. The positions of items 2 and 3 are changed in both π_1 and π_2.

Clearly this swap does not affect the number of chains, lengths of any chain, nor the occurrence frequencies of any itemset as items are not inserted or removed. To guarantee that also the $C_D(u,v)$s are preserved, we must pose one additional requirement for the swap. When transposing two adjacent items in the chain π_1, say, u and v with u originally before v, $C_D(u,v)$ is decremented by one as there is one instance less of u preceding v after the transposition, and $C_D(v,u)$ is incremented by one as now there is one instance more where v precedes u. Obviously, if the swap would change only π_1, the resulting data set would no longer belong to $\mathcal{C}(D)$ as $C_D(u,v)$ and $C_D(v,u)$ are changed. But the second transposition we carry out in π_2 cancels out the effect the first transposition had on $C_D(u,v)$ and $C_D(v,u)$, and the resulting set of chains remains in $\mathcal{C}(D)$.

Definition 7 *Let D be a set of chains and let π_1 and π_2 belong to D. The tuple (π_1, π_2, i, j) is a* valid swap *for D, if the item at the ith position of π_1 is the same as the item at the $j+1$th position of π_2, and if the item at $i+1$th position of π_1 is the same as the item at the jth position of π_2.*

The swap we show in the example above is thus a valid swap.

Given the data D, we may have several valid swaps to choose from. To see how the set of valid swaps evolves in a single step of the algorithm, consider the following example. Let D_i contain the three chains below:

$$\pi_1 : (1,2,3,4,5) \qquad \pi_2 : (7,8,4,3,6) \qquad \pi_3 : (3,2,6,4,1)$$

The valid swaps in this case are $(\pi_1, \pi_3, 2, 1)$ and $(\pi_1, \pi_2, 3, 3)$. If we apply the swap $(\pi_1, \pi_2, 3, 3)$ we obtain the chains

$$\pi_1' : (1,2,4,3,5) \qquad \pi_2' : (7,8,3,4,6) \qquad \pi_3 : (3,2,6,4,1)$$

Obviously $(\pi_1, \pi_2, 3, 3)$ is still a valid swap, as we can always revert the previous swap. But notice that $(\pi_1, \pi_2, 2, 1)$ is no longer a valid swap as the items 2 and 3 are not adjacent in π_1'. Instead $(\pi_2', \pi_3, 4, 3)$ is introduced as a new valid swap since now 4 and 6 are adjacent in π_2'.

Given this definition of the swap, is $\mathcal{C}(D)$ connected with respect to the valid swaps? Meaning, can we reach every member of $\mathcal{C}(D)$ starting from D? This is a desirable property as we want to sample uniformly from $\mathcal{C}(D)$, but so far this remains an open question.

4.3.3 CONVERGENCE

Above it was mentioned that we must let the Markov chain run long enough to make sure $\tilde{D}_s$ is not correlated with the starting state D_0. The chain should have *mixed*, meaning that when we stop it the probability of landing at a particular state D_s actually corresponds to the probability D_s has in the stationary distribution of the chain. Determining when a simulated Markov chain has converged to its stationary distribution is not easy.

Hence we resort to a fairly simple heuristic. An indicator of the current sample D_i being uncorrelated to $D_0 = D$ is the following measure:

$$\delta(D, D_i) = |D|^{-1} \sum_{j=1}^{|D|} d_K(D(j), D_i(j)), \tag{15}$$

where $D(j)$ is the jth chain in D. Note that $\delta(D, D_i)$ is always defined, as the chain $D_i(j)$ is a permutation of $D(j)$. The distance defined in Equation 15 is thus the average Kendall distance between the permutations in D and D_i. To assess the convergence we observe how $\delta(D, D_i)$ behaves as i grows. When $\delta(D, D_i)$ has converged to some value or is not increasing only at a very low rate, we assume the current sample is not correlated with D_0 more strongly than with most other members of $\mathcal{C}(D)$.

Note that here we are assuming that the chains in D are *labeled*. To see what this means consider the following example with the sets D and D_i both containing four chains.

$D(1) : 1, 2, 3$	$D_i(1) : 2, 1, 3$
$D(2) : 4, 5, 6$	$D_i(2) : 6, 5, 4$
$D(3) : 2, 1, 3$	$D_i(3) : 1, 2, 3$
$D(4) : 6, 5, 4$	$D_i(4) : 4, 5, 6$

Here we have obtained D_i from D with the multiple swap operations. The distance $\delta(D, D_i)$ is 2 even though D and D_i clearly are identical as sets. Hence, the measure of Equation 15 can not be used for testing this identity. To do this we should compute the Kendall distance between $D(j)$ and $D_i(h(j))$, where h is a bijective mapping between chains in D and D_i that minimizes the sum of the pairwise distances. However, we consider this simple approach sufficient for the purposes of this paper.

4.3.4 IMPLEMENTATION ISSUES

Until now we have discussed the approach at a general level. There's also a practical issue when implementing the proposed algorithm. The number of valid swaps at a given state is of order $O(m^2 n^2)$ in the worst case, which can get prohibitively large for storing each valid swap as a tuple explicitly. Hence, we do not store the tuples, but only maintain two sets that represent the entire set of swaps

but use a factor of n less space. We let

$$A_D = \{\{u,v\} \mid \exists \pi_1 \in D \text{st.} uv \in \pi_1 \wedge \exists \pi_2 \in D \text{st.} vu \in \pi_2\},$$

where $uv \in \pi$ denotes that u and v are adjacent in π with u before v. This is the set of *swappable pairs* of items. The size of A_D is of order $O(m^2)$ in the worst case. In addition, we also have the sets

$$S_D(u,v) = \{\pi \in D \mid uv \in \pi\}$$

for all (u,v) pairs. This is simply a list that contains the set of chains where we can transpose u and v. Note that $S_D(u,v)$ and $S_D(v,u)$ are not the same set. In $S_D(u,v)$ we have chains where u appears before v, while in $S_D(v,u)$ are chains where v appears before u. The size of each $S_D(u,v)$ is of order $O(n)$ in the worst case, and the storage requirement for A_D and S_D is hence only $O(m^2 n)$, a factor of n less than storing the tuples explicitly.

The sets A_D and S_D indeed fully represent all possible valid swaps. A valid swap is constructed from A_D and S_D by first picking a swappable pair $\{u,v\}$ from A_D, and then picking two chains, one from $S_D(u,v)$ and the other from $S_D(v,u)$. It is easy to see that a swap constructed this way must be a valid swap. Also, verifying that there are no valid swaps not described by A_D and S_D is straightforward.

There is still one concern. Recall that we want to use the Metropolis-Hastings approach to sample from the uniform distribution over $\mathcal{C}(D)$. In order to do this we must be able to sample uniformly from the neighbors of D_i, and we have to know the precise size of D_i's neighborhood. The size of the neighborhood $N(D_i)$ is precisely the number of valid swaps at D_i, and is given by

$$|N(D_i)| = \sum_{\{u,v\} \in A_{D_i}} |S_{D_i}(u,v)| \cdot |S_{D_i}(v,u)|,$$

which is easy to compute given A_{D_i} and S_{D_i}.

To sample a neighbor of D_i uniformly at random using A_{D_i} and S_{D_i}, we first pick the swappable pair $\{u,v\}$ from A_{D_i} with the probability

$$Pr(\{u,v\}) = \frac{|S_{D_i}(u,v)| \cdot |S_{D_i}(v,u)|}{|N(D_i)|}, \tag{16}$$

which is simply the fraction of valid swaps in $N(D_i)$ that affect items u and v. Then π_1 and π_2 are sampled uniformly from $S_D(u,v)$ and $S_D(v,u)$ with probabilities $|S_D(u,v)|^{-1}$ and $|S_D(v,u)|^{-1}$, respectively. Thus we have

$$Pr(\{u,v\}) \cdot |S_D(u,v)|^{-1} \cdot |S_D(v,u)|^{-1} = \frac{1}{|N(D_i)|}$$

as required.

The final algorithm that we call SWAP-PAIRS is given in Algorithm 2. It takes as arguments the data D and the integer s that specifies the number of rounds the algorithm is run. On lines 2–6 we initialize the sets A_D and S_D, while lines 8–20 contain the main loop. First, on line 9 the pair $\{u,v\}$ is sampled from A_D with the probability given in Equation 16. The SAMPLE-UNIFORM function simply samples an element from the set it is given as the argument. On lines 13 and 15 we compute the neighborhood sizes before and after the swap, respectively. The actual swap is carried out by the APPLY-SWAP function, that modifies π and τ in D and updates A_D and S_D accordingly. Lines 16–18 implement the Metropolis-Hastings step. Note that it is easier to simply perform the swap and backtrack if the jump should not have been accepted. A swap can be canceled simply by applying it a second time. The function RAND() returns a uniformly distributed number from the interval $[0,1]$.

Algorithm 2 The SWAP-PAIRS algorithm for sampling uniformly from $\mathcal{C}(D)$.

```
1: SWAP-PAIRS(D, s)
2: A_D ← {{u,v} | ∃π1 ∈ D st. uv ∈ π1 ∧ ∃π2 ∈ D st. vu ∈ π2}
3: for all {u,v} ∈ A_D do
4:     S_D(u,v) ← {π ∈ D | uv ∈ π}
5:     S_D(v,u) ← {π ∈ D | vu ∈ π}
6: end for
7: i ← 0
8: while i < n do
9:     {u,v} ← SAMPLE-PAIR(A_D, S_D)
10:    π ← SAMPLE-UNIFORM(S_D(u,v))
11:    τ ← SAMPLE-UNIFORM(S_D(v,u))
12:    s ← (π, τ, π(u), τ(v))
13:    N_before ← Σ_{{u,v}∈A_D} |S_D(u,v)| · |S_D(v,u)|
14:    APPLY-SWAP(s, D, A_D, S_D)
15:    N_after ← Σ_{{u,v}∈A_D} |S_D(u,v)| · |S_D(v,u)|
16:    if RAND() ≥ N_before / N_after then
17:        APPLY-SWAP(s, D, A_D, S_D)
18:    end if
19:    i ← i + 1
20: end while
21: return D
```

5. Experiments

In this section we discuss experiments that demonstrate how our algorithms perform on various artificial and real data sets. We consider a two-step algorithm that either starts with random initial clusters (RND), or a clustering that is computed with standard *k*-means (initialized with random centroids) in the graph (GR) or hypersphere (HS) representation. This initial clustering is subsequently refined with the variant of Lloyd's algorithm discussed in Section 2 to obtain the final clustering. We also compare our method against existing approaches by Kamishima and Akaho (2006). These algorithms, called TMSE and EBC, are similar clustering algorithms for sets of chains, but they are based on slightly different distance functions and types of centroid. We used original implementations of TMSE and EBC that were obtained from the authors.

5.1 Data Sets

The artificial data sets are generated by the procedure described in Section 3.1.1. In addition to artificial data we use four real data sets that are all based on publicly available sources. The data consist of preference rankings that are either explicit, derived, or observed. We say a preference ranking is *explicit* if the preferences are directly given as a ranked list of alternatives. A preference ranking is *derived* if the ranking is based on item-specific scores, such as movie ratings. Finally, a preference ranking is *observed* if it originates from a source where preferences over alternatives only manifest themselves indirectly in different types of behavior, such as web server access logs.

	SUSHI	MLENS	DUBLIN	MSNBC
n	5000	2191	5000	5000
m	100	207	12	17
min. l	10	6	4	6
avg. l	10	13.3	4.8	6.5
max. l	10	15	6	8

Table 1: Key statistics for different real data sets. The number of chains, the number of items, and the length of a chain are denoted by n, m, l, respectively.

Key statistics of the data sets are summarized in Table 1. More details are given below for each data set.

5.1.1 SUSHI

These data are explicit preference rankings of subsets of 100 items. Each chain is a response from a survey[2] where participants were asked to rank 10 flavors of sushi in order of preference. Each set of 10 flavors was chosen randomly from a total set of 100 flavors. The data consists of 5000 such responses.

5.1.2 MLENS

These data are derived preference rankings of subsets of 207 items. The original data consists of movie ratings (1–5 stars) collected by the GroupLens[3] research group at University of Minnesota. We discarded movies that had been ranked by fewer than 1000 users and were left with 207 movies. Next we pruned users who have not used the entire scale of five stars in their ratings and were left with 2191 users. We generate one chain per user by first sampling a subset of movies the user has rated, so that at most three movies having the same rating are in the sample. Finally we order the sample according to the ratings and break ties in ratings arbitrarily.

5.1.3 DUBLIN

These data are explicit preference rankings of subsets of 12 items. Each chain is a vote placed in the 2002 general elections in Ireland.[4] and ranks a subset of 12 candidates from the electoral district of northern Dublin. We only consider votes that rank at least 4 and at most 6 candidates and are left with 17737 chains. Of this we took a random sample of 5000 chains for the analysis.

5.1.4 MSNBC

These data are observed preference rankings over 17 items. Each chain shows the order in which a user accessed a subset of 17 different sections of a web site (`msnbc.com`).[5] Each chain contains only the first occurrence of a category, subsequent occurrences were removed. Also, we selected a

2. The SUSHI data be found at `http://www.kamishima.net/sushi` (29 April 2011).
3. The MLENS data can be found at `http://www.grouplens.org/node/12` (29 April 2011).
4. At the time of publication this data can be found by accessing old versions of `http://www.dublincountyreturningofficer.com/` in the Internet Archive at `http://waybackmachine.org`.
5. MSNBC data can be found at `http://kdd.ics.uci.edu/databases/msnbc/` (29 April 2011).

subset of the users who had visited at least 6 and at most 8 different categories and were left with 14598 chains. Again we used a random subset of 5000 chains for the analysis.

5.2 Recovering a Planted Clustering

In this section we discuss experiments on artificial data, with the emphasis on studying the performance of the algorithms under different conditions. These conditions can be characterized by parameters of the input data, such as length of the chains or total number of items. The task is to recover a "true" clustering that was planted in the input data.

5.2.1 EXPERIMENTAL SETUP

The notion of correctness is difficult to define when it comes to clustering models. With real data we do not in general know the correct structure, or if there even is any structure to be found. To have a meaningful definition of a correct clustering, we generate synthetic data that contains a planted clustering. We compare this with the clusterings found by the algorithms.

To measure the similarity between two clusterings we use a variant of the Rand Index (Rand, 1971) called the Adjusted Rand Index (Lawrence and Phipps, 1985). The basic Rand Index essentially counts the number of pairs of points where two clusterings agree (either both assign the points in the same cluster, or both assign the points in different clusters), normalized by the total number of pairs. The maximum value for two completely agreeing clusterings is thus 1. The downside with this approach is that as the number of clusters increases, even random partitions will have a score close to 1, which makes it difficult to compare algorithms. The Adjusted Rand Index corrects for this by normalizing the scores with respect to the expected value of the score under the assumption that the random partition follows a generalized hypergeometric distribution (Lawrence and Phipps, 1985).

Artificial sets of chains are created with the procedure described in Section 3.1.1. Instead of arbitrary partial orders as the components, we use bucket orders (or *ordered partitions*) of M. More specifically, a bucket order on M is a totally ordered set of disjoint subsets (buckets) of M that cover all items in M. If the items u and v both belong to the bucket $M_i \subseteq M$, they are unordered. If $u \in M_i \subseteq M$ and $v \in M_j \subseteq M$, and M_i precedes M_j, then also u precedes v. We used bucket orders with 10 buckets in the experiments.

Input size n is fixed to 2000. We varied the following parameters: length of a chain l, total number of items m, and number of clusters in the true clustering κ. We ran the algorithms on various combinations of these with different values of k, that is, we also wanted to study how the algorithms behave when the correct number of clusters is not known in advance.

5.2.2 COMPARING INITIALIZATION STRATEGIES

Results for our variant of Lloyd's algorithm with the three different initialization strategies (HS, GR, and RND) are shown in Figure 1 for a number of combinations of k and m. Here we only plot cases where $k = \kappa$, meaning that the algorithm was given the correct number of clusters in advance. The grey lines are 95 percent confidence intervals. As on one hand suggested by intuition, and on the other hand by Theorem 2, finding a planted clustering becomes easier as the length of the chains increase. With $l = 9$ the original clustering is found almost always independent of the values of m and k. For smaller values of l the effect of m and k is stronger. The problem becomes more difficult as m and k increase. When comparing the initialization strategies, HS and GR outperform RND.

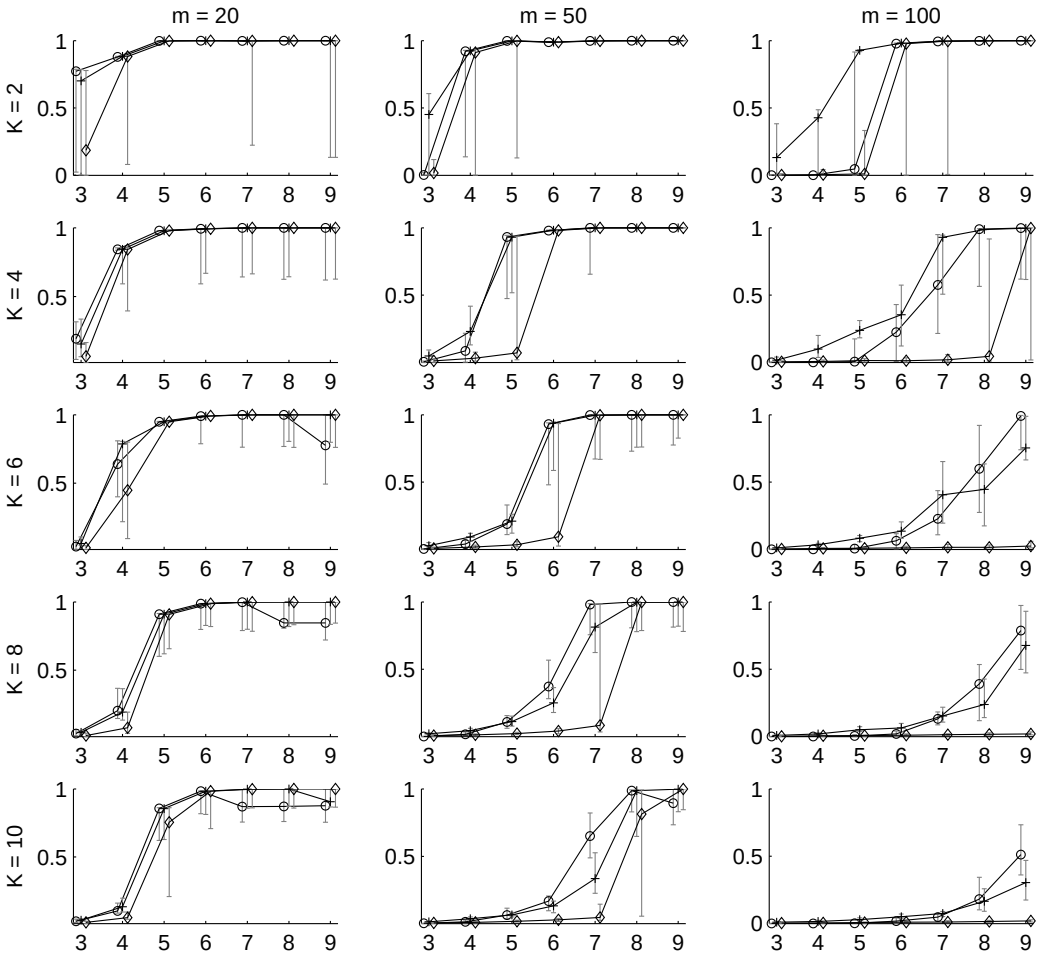

Figure 1: The Adjusted Rand Index (median over 25 trials) between a recovered clustering and the true clustering as a function of the length of a chain in random data sets consisting of 2000 chains each. Initialization methods are ○: GR, +: HS, and ◇: RND. Gray lines indicate 95 percent confidence intervals.

5.2.3 COMPARING AGAINST EXISTING METHODS

We compared how our approach using the HS initialization compares with existing algorithms. The HS-based variant was chosen because of fairness: The process we use to generate artificial data exactly matches the assumption underlying the GR approach, and hence may give this algorithm an unfair advantage. Also, the HS initialization is faster to compute.

Results are shown in Figure 2 for $m = 10$ and $m = 100$, and $k \in \{2, 6, 10\}$. The total number of items m has a strong effect on the performance. As above, the problem or recovering the clustering becomes harder as m increases and l decreases. Our algorithm suffers from very poor performance

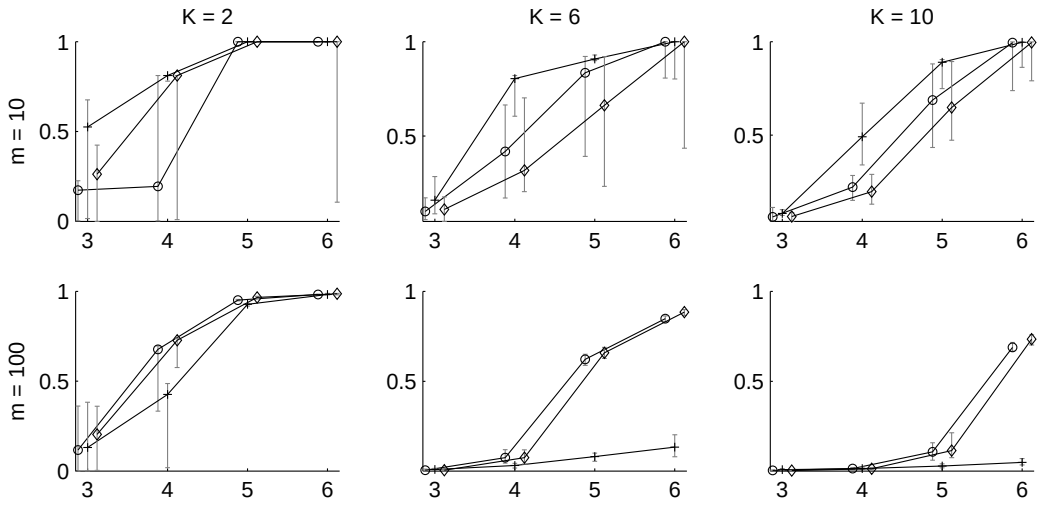

Figure 2: The Adjusted Rand Index (median over 25 trials) between a recovered clustering and the true clustering as a function of the length of a chain. Labels are: $+$: our algorithm initialized using HS, $\circ$: EBC, $\diamond$: TMSE.

with $m = 100$, while the EBC and TMSE algorithms can recover the planted clustering rather well also in this case. In contrast, for $m = 10$ and small l, our approach yields better results especially for $k > 2$. Recall that our algorithm relies on the pairwise probabilities of one item to precede an other. When $m = 100$ we have 4950 distinct pairs of items, when $m = 10$ this number is merely 45. With a large m it is therefore likely that our estimates of the pairwise probabilities are noisy simply because there are less observations of individual pairs since the input size is fixed. By increasing the size of the input these estimates should become more accurate.

We tested this hypothesis by running an experiment with random data sets ten times larger, that is, with an input of 20000 chains on $m = 100$ items. We concentrated on two cases: $k = 2$ with $l = 4$, and $k = 6$ with $l = 6$. The first corresponds to a situation where there is a small gap between the performance of TMSE/EBC and our method, and all algorithms show mediocre performance (see Fig. 2, 2nd row, left column). The second combination of k and l covers a case where this gap is considerably bigger, and TMSE/EBC both do rather well in recovering the planted clustering (see Fig. 2, 2nd row, middle column). Results are shown in Table 2. Increasing the size of the input leads to a considerable increase in performance of our algorithm. This suggests that for large data sets the approach based on pairwise probabilities may yield results superior to those obtained with existing algorithms.

5.2.4 UNKNOWN SIZE OF TRUE CLUSTERING

So far we have only considered cases where $k = \kappa$, that is, the algorithms were given the correct number of clusters. When analyzing real data κ is obviously unknown. We studied the algorithms' sensitivity to the value of k. Figure 3 shows the Adjusted Rand Index for our algorithm with HS initialization, and the EBC and TMSE algorithms when $m = 20$, and $\kappa = 6$. All three algorithms

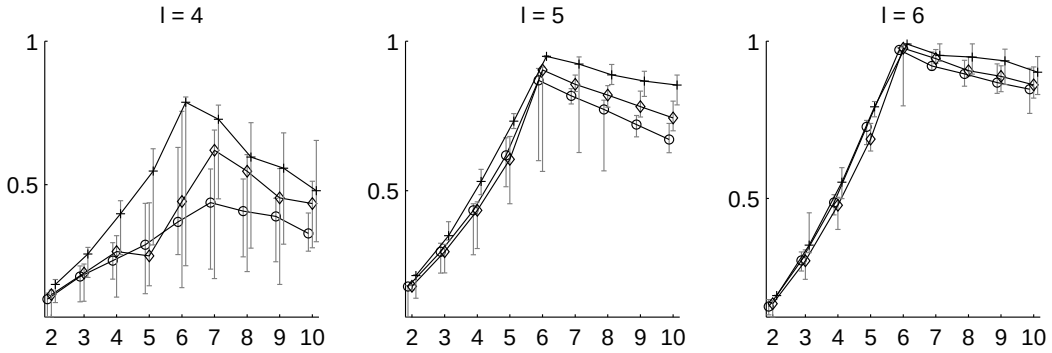

Figure 3: Adjusted Rand Index (median over 25 trials) as a function of k for different algorithms. We have $\kappa = 6$, and $m = 20$ in each case, the value of l is shown above each curve. Labels are: $+$: our algorithm initialized using HS, $\circ$: EBC, $\diamond$: TMSE.

perform similarly. For short chains ($l = 4$) the differences are somewhat stronger. While our HS-based method seems to be marginally better in recovering the original clustering, there is a lot of overlap in the confidence intervals, and none of the algorithms is able to find the true clustering exactly. We also acknowledge that the stronger performance of our method with $l = 5$ and $l = 6$ may be attributed to an implementation detail: Our algorithm is not guaranteed to return k clusters, it may return a number less than k if one of the clusters becomes empty during the computation. It is not clear how the implementations of TMSE and EBC deal with empty clusters.

5.3 Experiments with Real Data

This experiment was carried out by computing a k-way clustering of each data set described in Section 5.1 with k ranging from 2 to 10. Performance is measured by the clustering error as defined in Equation 1, using the centroid and distance function that are described in Section 2.3. Each combination of algorithm, data, and k was repeated 25 times with a randomly chosen initial clustering. (Note that even if we initialize our method by computing a clustering using either of the vector space representations, the algorithms that compute these must be initialized somehow.)

Figure 4 shows the reconstruction error as a function of k. Note that values on the y-axis have been normalized by the baseline error of having all chains in the same cluster. The error bars indicate 95 percent confidence intervals. The EBC algorithm is omitted from the figures, as this method was consistently outperformed by the TMSE algorithm. This result is also in line with previous empirical

	EBC		TMSE		HS init.	
$k = 2, l = 4$	0.817	(0.816, 0.822)	0.818	(0.816, 0.822)	**0.891**	(0.891, 0.892)
$k = 6, l = 6$	0.935	(0.932, 0.938)	0.937	(0.934, 0.939)	**0.974**	(0.973, 0.976)

Table 2: Adjusted Rand Index (median over 25 trials) for different methods computed from artificial data consisting of 20000 chains with $m = 100$ and the shown values for k and l. Numbers in parenthesis indicate 95 percent confidence intervals.

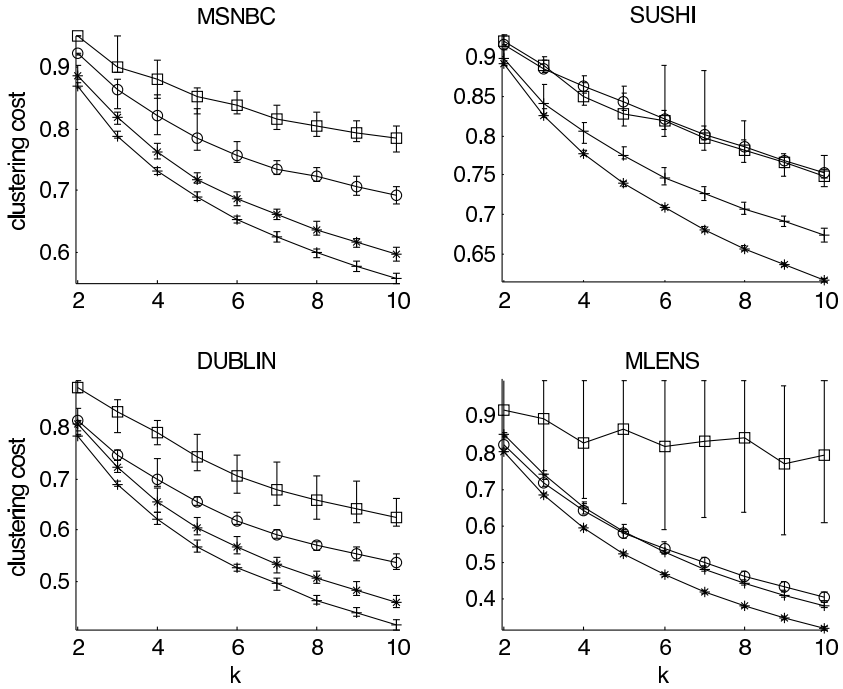

Figure 4: Reconstruction error as expressed in Equation 1 with the definitions of distance and centroid from Section 2.3 as a function of k. The y-axis has been normalized to show the error as a fraction of the baseline error of $k = 1$. Legend: $\square$: *only* standard k-means in GR representation, $\circ$: *only* standard k-means in HS representation, $+$: our variant of Lloyd's algorithm with RND init., $*$: the TMSE algorithm.

evidence reported by Kamishima and Akaho (2006). We also left out results obtained with our algorithm using either of the vector space representations to compute an initial clustering. (The curves for HS and GR therefore show performance that is obtained simply by mapping chains to the respective vector spaces and running standard k-means.) Contrary to random data (see results of Section 5.2), these initialization strategies did not give significantly better results than simple random initialization.

Our k-means procedure outperforms the TMSE algorithm with the MSNBC and DUBLIN data sets. With SUSHI and MLENS the situation is reversed. This statement holds for all values of k, and seems robust as the confidence intervals do not overlap. Also, when measuring clustering quality in this way, the results obtained by using only the vector space representations are considerably inferior to the other methods. Of course this is not an entirely fair comparison as the objective functions differ. In Figure 5 we plot the reconstruction error computed with the distance function and centroid representation used by TMSE. For details, please see Kamishima and Akaho (2006, Section 3.1). Using this measure, the SUSHI and MLENS data sets demonstrate an even stronger difference between the methods. With MSNBC and DUBLIN our algorithm continues to perform somewhat better, albeit this time the confidence intervals overlap. Interestingly, if an algorithm is better, it is better independent of the cost function used to evaluate the result. For instance, with MSNBC and DUBLIN our algorithm marginally outperforms TMSE even in terms of TMSE's own

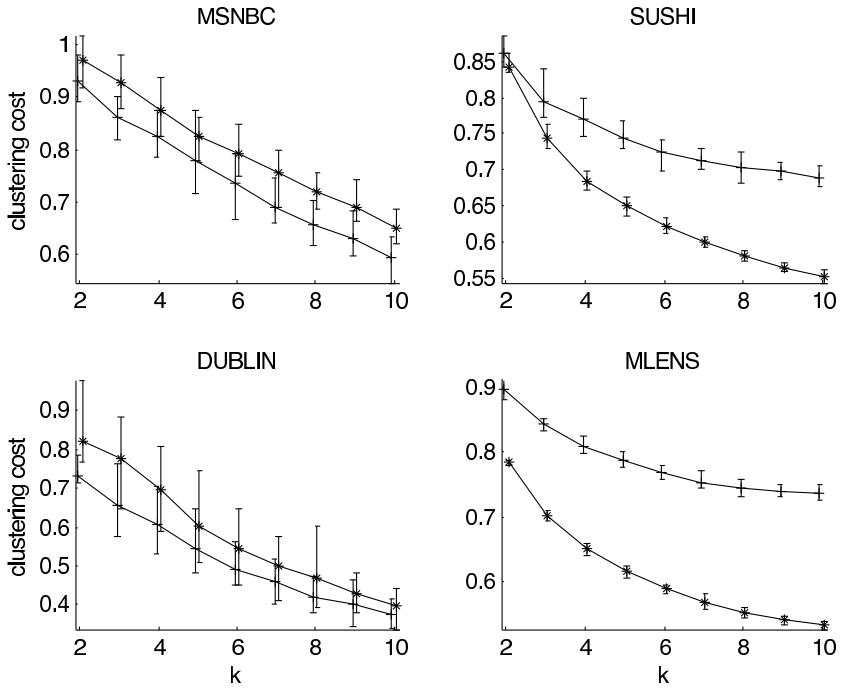

Figure 5: Reconstruction error of a clustering as expressed in Equation 1 with the definitions of distance and centroid from Kamishima and Akaho (2006) as a fraction of the baseline error of $k = 1$. Legend: $+$: our algorithm with random initialization, $*$: the TMSE algorithm.

cost function, and vice versa with SUSHI and MLENS. These results are in line with the ones we obtain with artificial data. As can be seen in Table 1, the data sets MSNBC and DUBLIN have a considerably smaller m. The experiments in Section 5.2.3 suggest that by collecting more data we could improve our result for the SUSHI and MLENS data sets.

5.4 Testing Clustering Validity

We use the randomization method of Section 4 to test the interestingness of the found clusterings. A clustering is assumed to be interesting if its test statistic substantially differs from the ones we obtain from randomized data. The test statistic we use is the reconstruction error given in Equation 1. The methods use their respective definitions of distance and centroid to compute the error.

To carry out the test, we must first estimate how many swaps are needed to obtain a single sample that is uncorrelated with the original data. To this end we run the SWAP-PAIRS algorithm for 10×10^6 swaps on each data set and measure $\delta(D, D_i)$ every 0.1×10^6 swaps. The assumption is that the data D_i are uncorrelated with the initial state D when $\delta(D, D_i)$ no longer increases. Figure 6 shows how the distance $\delta(D, D_i)$ develops with the number of swaps i for the data sets. From this we can read the number of swaps that are needed to obtain approximately uncorrelated samples. For SUSHI and MLENS the Markov chain seems to converge after approximately 5×10^6 swaps, for DUBLIN the distance $\delta(D, D_i)$ stabilizes already after about 0.5×10^6 swaps, while with MSNBC this happens after roughly 3×10^6 swaps. The randomized data used in the remaining analysis are

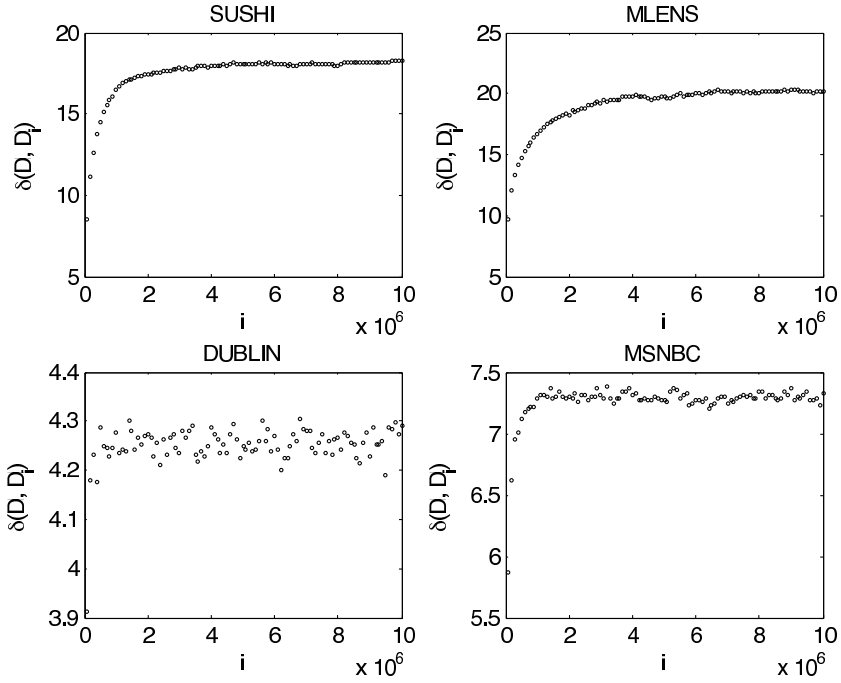

Figure 6: The distance $\delta(D, D_i)$ as a function of the number of swaps i.

computed using these swap counts, respectively. Here we also want to point out that randomization is computationally intensive. The table below shows the times to perform 10×10^6 swaps for the different data sets.

	SUSHI	MLENS	DUBLIN	MSNBC
t (seconds)	297	670	73	81

We observe that n, the number of chains in the input, does not affect the running time t, but the number of items m plays a significant part. (See also Table 1.)

For the actual analysis we sample 99 random instances from the equivalence class of each data set, and compare the test statistic with the one obtained from real data. Figure 7 shows the histogram of the reconstruction error in randomized data together with the minimum, maximum, and median error over 25 trials with real data. If this interval is clearly to the left of the histogram, it is unlikely to observe an error of the same magnitude in randomized data. If the interval overlaps with the histogram, the results should be considered as not significant according to this test.

In general the results suggest that the clusterings we obtain from the actual data sets have a smaller reconstruction error than a clustering computed with the same algorithm from a randomized data. There are some interesting exceptions, however. For MSNBC, SUSHI, and DUBLIN the clusterings obtained by our method from real data seem considerably better than those we obtain in random data, independent of K. In case of MLENS the results are clearly not significant for any K. For the TMSE algorithm the test suggests a significant outcome in case of SUSHI and MLENS, while for MSNBC and DUBLIN the clustering from real data is not considerably better than a clustering from randomized data.

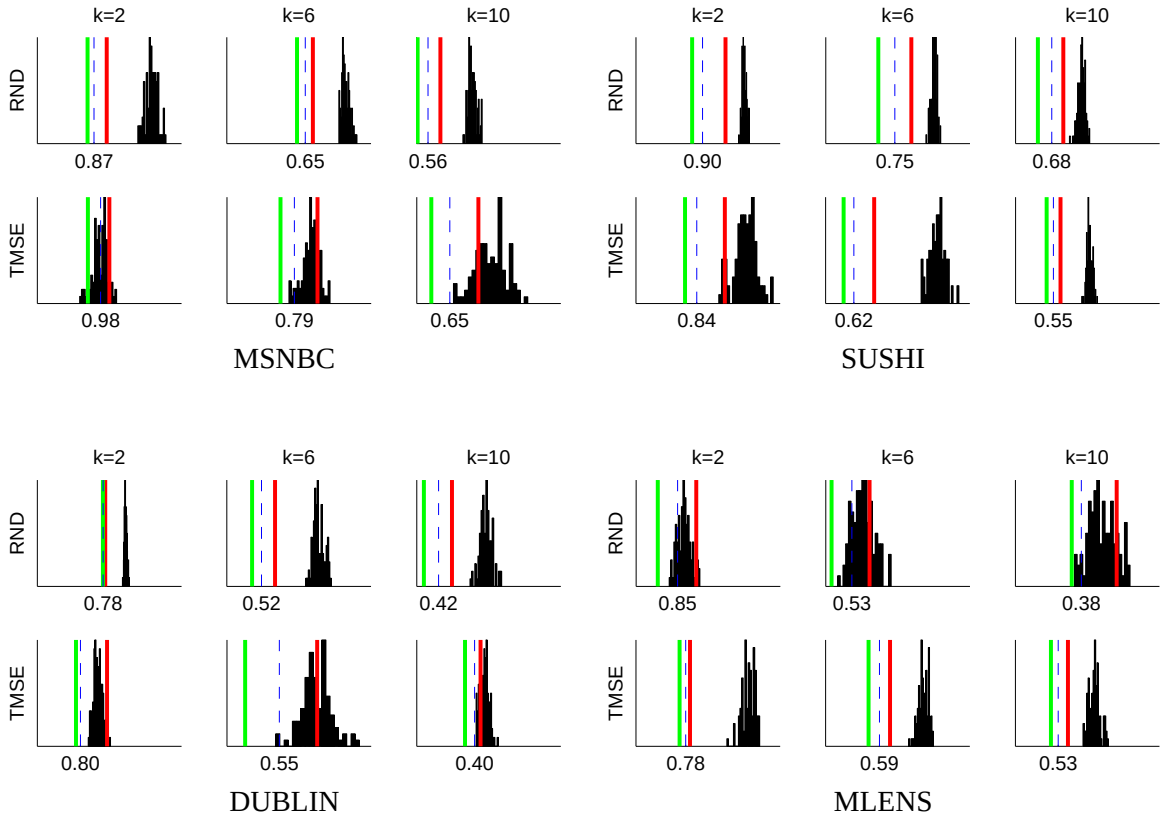

Figure 7: Results of randomization testing with our algorithm using RND initialization and the TMSE algorithm for different values of K. The numbers are normalized by the clustering error for $K = 1$. The histograms show the distribution of the clustering error on the randomized data. The light gray (green online), dashed, and dark gray (red online) lines indicate the minimum, median, and maximum of the clustering error on the original data. Both algorithms use their own cost functions.

6. Conclusion

We have discussed the problem of clustering chains. First, in Section 2 we gave simple definitions of a centroid and a distance function that can be used together with Lloyd's algorithm (k-means) for computing a clustering directly using chains. In Section 3 we gave two methods for mapping chains to a high-dimensional vector space. These representations have the advantage that any clustering algorithm can be used. Moreover, a clustering obtained in this way can still be further refined using the technique of Section 2. Mapping chains to vector spaces is an interesting subject in its own right and can have many other uses in addition to clustering. For example, they can be used to visualize of sets of chains, as was done by Ukkonen (2007), as well as by Kidwell et al. (2008). Also, we believe that the connections to the planted partition model (Condon and Karp, 2001; Shamir and Tsur, 2002) are very interesting at least from a theoretical point of view.

We also proposed a method for testing if a clustering found in a set of chains is any different from a clustering of random data. If the value of a suitable test statistic, such as the reconstruction error, does not substantially differ between the original input and the randomized data sets the clustering found in real data is probably not very meaningful. To this end we devised an MCMC algorithm for sampling sets of chains that all belong to the same equivalence class as a given set of chains.

In the experiments we compared our methods with the TMSE and EBC algorithms by Kamishima and Akaho (2009). We observe that for some data sets our algorithm yields better results, while for some other data sets the TMSE algorithm is a preferred choice. Interestingly, these differences can also be seen in the randomization tests. When an algorithm performs poorly, the results tend to be not significant according to the randomization test. Moreover, it seems that in cases where the TMSE algorithm is superior, our algorithm does not have enough data to perform well. Experiments on artificial data indicate that as the size of the input is increased (and other variables left unchanged), the performance of our algorithm increases considerably, and even outperforms the TMSE algorithm. Therefore, we suspect that by increasing data size we could improve the performance of our algorithm also with real data.

The main difference between the algorithms is the notion of distance. TMSE essentially uses a modified version of Spearman's rank correlation coefficient that is a "positional" distance for permutations, as it only considers the positions in which different items appear. We propose a "pairwise" distance that considers how pairs of items are related to each other. The experiments suggest that the pairwise approach is more powerful as long as there is enough data, but for smaller data sets positional distances seem more robust. Finding the tipping point in terms of input size and other data parameters where the pairwise approach becomes favorable over positional distances is an interesting open question.

Acknowledgments

I would like to thank the anonymous reviewers for their valuable feedback that helped to improve this manuscript considerably. This work was partly funded by the Academy of Finland (grant 131276).

Appendix A. Proofs of Theorems

Proofs of Theorem 2 and Theorem 5 are given below.

A.1 Proof of Theorem 2

The proof is a simple matter of upper bounding Equation 6. First we note that using Vandermonde's convolution (Graham et al., 1994, Equation 5.22) the sum in Equation 6 can be rewritten as

$$\binom{m}{l} - \Big(\underbrace{\binom{l}{1}\binom{m-l}{l-1} + \binom{m-l}{l}}_{A}\Big).$$

Essentially Vandermonde's convolution states that $\sum_{i=0}^{l}\binom{l}{i}\binom{m-l}{l-i} = \binom{m}{l}$, and we simply subtract the first two terms indicated by A, because above the sum starts from $i = 2$. Using simple manipulations

we obtain

$$A = \binom{m-l}{l}\Big(\frac{l^2}{m-2l+1}+1\Big),$$

which gives the following:

$$p = \binom{m}{l}^{-1}\Big(\binom{m}{l} - \binom{m-l}{l}\Big(\frac{l^2}{m-2l+1}+1\Big)\Big).$$

With $l < m/2$ the part $\frac{l^2}{m-2l+1}+1$ is lower bounded by 1, and we have

$$\begin{aligned}
p &< \binom{m}{l}^{-1}\Big(\binom{m}{l} - \binom{m-l}{l}\Big) = 1 - \binom{m}{l}^{-1}\binom{m-l}{l} \\
&= 1 - \frac{(m-l)!}{l!(m-2l)!}\cdot\frac{l!(m-l)!}{m!} \\
&= 1 - \frac{(m-l)(m-l-1)\cdots(m-2l+1)}{m(m-1)\cdots(m-l+1)} \\
&< 1 - \frac{(m-l)(m-l-1)\cdots(m-2l+1)}{m^l} \\
&< 1 - \frac{(m-2l+1)^l}{m^l} < \frac{m^l-(m-2l)^l}{m^l}.
\end{aligned}$$

We can factor $m^l-(m-2l)^l$ as follows:

$$\begin{aligned}
m^l-(m-2l)^l &= (m-(m-2l))\Big(m^{l-1}(m-2l)^0 + m^{l-2}(m-2l)^1 + \ldots \\
&\qquad \cdots + m^1(m-2l)^{l-2} + m^0(m-2l)^{l-1}\Big) \\
&= 2l\sum_{i=0}^{l-1} m^{l-1-i}(m-2l)^i.
\end{aligned}$$

Using this we write

$$\frac{m^l-(m-2l)^l}{m^l} = 2l\sum_{i=0}^{l-1}(\frac{1}{m})^l m^{l-1-i}(m-2l)^i.$$

Letting $a = l-1$ and taking one $\frac{1}{m}$ out of the sum we get

$$\begin{aligned}
\frac{1}{m}2(a+1)\sum_{i=0}^{a}(\frac{1}{m})^a m^{a-i}(m-2(a+1))^i &= \frac{1}{m}2(a+1)\sum_{i=0}^{a}(\frac{1}{m})^i(m-2(a+1))^i \\
&= \frac{1}{m}2(a+1)\sum_{i=0}^{a}(1-\frac{2(a+1)}{m})^i.
\end{aligned}$$

We assume $l = a+1$ is considerably smaller than m, and hence $(1-\frac{2(a+1)}{m})^i$ is at most 1. There are $a+1$ terms in the sum, so the above is upper bounded by $\frac{1}{m}2(a+1)(a+1) = 2\frac{l^2}{m}$, which concludes the proof of the theorem.

A.2 Proof of Theorem 5

Let $u \in \pi$: We start by showing that the claim of Equation 12 holds for all u that belong to π. That is, we will show that

$$\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = Q\big(-\frac{|\pi|+1}{2} + \pi(u)\big) \tag{17}$$

for all $u \in \pi$. First, note that $\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u)$ can be rewritten as follows

$$\sum_{\tau \in \mathcal{E}(\pi)} -\frac{m+1}{2} + \tau(u) = \sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} \#\{\tau(u) = i\}\Big(-\frac{m+1}{2} + i\Big), \tag{18}$$

where $\#\{\tau(u) = i\}$ denotes the number of times u appears at position i in the linear extensions of π. The sum is taken over the range $\pi(u), \ldots, m - |\pi| + \pi(u)$, as $\tau(u)$ can not be less than $\pi(u)$, because the items that appear before u in π must appear before it in τ as well, likewise for the other end of the range.

To see what $\#\{\tau(u) = i\}$ is, consider how a linear extension τ of π is structured. When u appears at position i in τ, there are exactly $\pi(u) - 1$ items belonging to π that appear in the $i-1$ indices to the left of u, and $|\pi| - \pi(u)$ items also belonging to π that appear in the $m - i$ indices to the right of u. The ones on the left may choose their indices in $\binom{i-1}{\pi(u)-1}$ different ways, while the ones on the right may choose their indices in $\binom{m-i}{|\pi|-\pi(u)}$ different ways. The remaining items that do not belong to π are assigned in an arbitrary fashion to the remaining $m - |\pi|$ indices. We have thus,

$$\#\{\tau(u) = i\} = \binom{i-1}{\pi(u)-1}\binom{m-i}{|\pi|-\pi(u)}(m-|\pi|)!.$$

When this is substituted into the right side of (18), and after rearranging the terms slightly, we get

$$\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = (m-|\pi|)! \sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} \binom{i-1}{\pi(u)-1}\binom{m-i}{|\pi|-\pi(u)}\Big(-\frac{m+1}{2} + i\Big).$$

This can be written as

$$\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = (m-|\pi|)!(S_1 + S_2), \tag{19}$$

where

$$\begin{aligned} S_1 &= -\frac{m+1}{2} \sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} \binom{i-1}{\pi(u)-1}\binom{m-i}{|\pi|-\pi(u)}, \quad \text{and} \\ S_2 &= \sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} i\binom{i-1}{\pi(u)-1}\binom{m-i}{|\pi|-\pi(u)}. \end{aligned}$$

Let us first look at S_2. The part $i\binom{i-1}{\pi(u)-1}$ can be rewritten as follows:

$$i\binom{i-1}{\pi(u)-1} = \frac{i\ (i-1)!}{(\pi(u)-1)!(i-\pi(u))!} \cdot \frac{\pi(u)}{\pi(u)} = \pi(u)\frac{i!}{\pi(u)!(i-\pi(u))!} = \pi(u)\binom{i}{\pi(u)}.$$

This gives

$$S_2 = \pi(u) \sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} \binom{i}{\pi(u)} \binom{m-i}{|\pi|-\pi(u)} = \pi(u) \binom{m+1}{|\pi|+1},$$

where the second equality is based on Equation 5.26 in Graham et al. (1994). Next we must show that $\binom{m+1}{|\pi|+1}$ will appear in S_1 as well. We can rewrite the sum as follows:

$$\sum_{i=\pi(u)}^{m-|\pi|+\pi(u)} \binom{i-1}{\pi(u)-1} \binom{m-i}{|\pi|-\pi(u)} = \sum_{i=\pi(u)-1}^{m-|\pi|+\pi(u)-1} \binom{i}{q} \binom{r-i}{p-q},$$

where $q = \pi(u)-1$, $r = m-1$ and $p = |\pi|-1$. Again we apply Equation 5.26 of Graham et al. (1994) to get

$$S_1 = -\frac{m+1}{2} \binom{r+1}{p+1} = -\frac{m+1}{2} \binom{m}{|\pi|},$$

which we multiply by $\frac{|\pi|+1}{|\pi|+1}$ and have

$$S_1 = -\frac{|\pi|+1}{2} \cdot \frac{m+1}{|\pi|+1} \binom{m}{|\pi|} = -\frac{|\pi|+1}{2} \binom{m+1}{|\pi|+1}.$$

When S_1 and S_2 are substituted into (19) we have

$$\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = (m-|\pi|)! \Big(-\frac{|\pi|+1}{2} \binom{m+1}{|\pi|+1} + \pi(u) \binom{m+1}{|\pi|+1} \Big),$$

which is precisely Equation 17 when we let $Q = (m-|\pi|)! \binom{m+1}{|\pi|+1}$.

Let $u \notin \pi$: To complete the proof we must still show that Equation 12 also holds for items u that do not appear in the chain π. For such u we have $f_\pi(u) = 0$ by definition. Since we showed above that $Q > 0$, we have to show that $\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = 0$ when $u \notin \pi$ to prove the claim.

We'll partition $\mathcal{E}(\pi)$ to disjoint groups defined by index sets I. Let $S(I)$ denote the subset of $\mathcal{E}(\pi)$ where the items that belong to π appear at indices $I = \{i_1, \ldots, i_{|\pi|}\}$. Furthermore, let $I^R = \{m - i_1 + 1, \ldots, m - i_{|\pi|} + 1\}$. See Figure 8 for an illustration of the structure of the permutations that belong to $S(I)$ and $S(I^R)$.

Now we can write for every $u \notin \pi$:

$$\sum_{\tau \in \mathcal{E}(\pi)} f_\tau(u) = \frac{1}{2} \sum_I \sum_{\tau \in \{S(I) \cup S(I^R)\}} f_\tau(u). \tag{20}$$

That is, we first sum over all possible index sets I, and then sum over all τ that belong to the union of $S(I)$ and $S(I^R)$. Each I is counted twice (once as I and once as I^R), so we multiply the right hand side by $\frac{1}{2}$. To make sure that Equation 20 equals zero, it is enough to show that $\sum_{\tau \in \{S(I) \cup S(I^R)\}} f_\tau(u) = 0$ for each I.

Note that we have $f_\tau(u) + f_{\tau^R}(u) = 0$ because $\tau^R(u) = m - \tau(u) + 1$. That is, the values at indices j and $m-j+1$ cancel each other out. This property will give us the desired result if we can show that for each permutation $\tau \in \{S(I) \cup S(I^R)\}$ where an item $u \notin \pi$ appears at position j, there exists a corresponding permutation τ', also in $\{S(I) \cup S(I^R)\}$, where u appears at position $m-j+1$. Denote by $\#(S,u,j)$ the size of the set $\{\tau \in S \mid \tau(u) = j\}$.

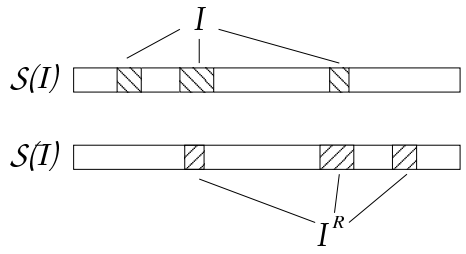

Figure 8: Permutations in $S(I)$ have the positions I occupied by items that belong to the chain π, while permutations in $S(I^R)$ have the positions I^R occupied by items of π. See proof of Theorem 5.

An index is *free* if it *does not* belong to the set $\{I \cup I^R\}$. Let j be a free index. By definition of the sets I and I^R, $m-j+1$ is also a free index. We have $\#(S(I),u,j) = \#(S(I),u,m-j+1)$. This holds for $S(I^R)$ as well. As a consequence, when we sum over all permutations in $\{S(I) \cup S(I^R)\}$, the values corresponding to index j and $m-j+1$ cancel each other out because u appears equally many times at positions j and $m-j+1$. The total contribution to the sum $\sum_{\tau \in \{S(I) \cup S(I^R)\}} f_\tau(u)$ of u appearing at the free indices is therefore zero.

Let j belong to I, meaning it is not free. By definition of the sets I and I^R, the index $m-j+1$ now belongs to I^R, and is also not free. However, because of symmetry we have $\#(S(I^R),u,j) = \#(S(I),u,m-j+1)$. That is, the number of times the item u appears at position j in a permutation belonging to $S(I^R)$ is the same as the number of times it appears at position $m-j+1$ in a permutation belonging to $S(I)$. When we sum over the permutations in $\{S(I) \cup S(I^R)\}$, the values of u appearing at position j in $S(I^R)$ are cancelled out by the values of u appearing at position $m-j+1$ in $S(I)$. The total contribution to the sum $\sum_{\tau \in \{S(I) \cup S(I^R)\}} f_\tau(u)$ of u appearing at an index in I is therefore zero as well. This concludes the proof of Theorem 5.

References

N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 684–693, 2005.

E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

D Arthur and S Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

G. H. Ball and D. J. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155, 1967.

A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 49–57, 2002.

P Berkhin. *Grouping Multidimensional Data*, chapter A Survey of Clustering Data Mining Techniques, pages 25–71. Springer, 2006.

J. Besag and P. Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76(4):633–642, 1989.

J. Besag and P. Clifford. Sequential Monte Carlo *p*-values. *Biometrika*, 78(2):301–304, 1991.

L. M. Busse, P. Orbanz, and J. M. Buhmann. Cluster analysis of heterogeneous rank data. In *Proceedings of the 24th international conference on Machine learning*, pages 113–120, 2007.

S Clémençon and J Jakubowicz. Kantorovich distances between rankings with applications to rank aggregation. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010*, 2010.

A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.

D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 776–782, 2006.

D. Critchlow. *Metric Methods for Analyzing Partially Ranked Data*, volume 34 of *Lecture Notes in Statistics*. Springer-Verlag, 1985.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference*, 2001.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Texts in Statistical Science. Chapman & Hall, CRC, 2004.

A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data*, 1(3), 2007.

P I Good. *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*, volume 2 of *Springer series in statistics*. Springer, 2000.

R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994.

D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. The MIT Press, 2001.

T. Kamishima and S. Akaho. Efficient clustering for orders. In *Workshops Proceedings of the 6th IEEE International Conference on Data Mining*, pages 274–278, 2006.

T. Kamishima and S. Akaho. *Mining Complex Data*, volume 165 of *Studies in Computational Intelligence*, chapter Efficient Clustering for Orders, pages 261–279. Springer, 2009.

T. Kamishima and J. Fujiki. Clustering orders. In *Proceedings of the 6th International Conference on Discovery Science*, pages 194–207, 2003.

P Kidwell, G Lebanon, and W S Cleveland. Visualizing incomplete and partially ranked data. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1356–1363, 2008.

H Lawrence and A Phipps. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982.

H. Moulin. *Axioms of Cooperative Decision Making*. Cambride Universiy Press, 1991.

T. B. Murphy and D. Martin. Mixtures of distance-based models for ranking data. *Computational Statistics & Data Analysis*, 41:645–655, 2003.

W M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

R. Shamir and D. Tsur. Improved algorithms for the random cluster graph model. In *Proceedings of Scandanavian Workshop on Algorithms Theory*, pages 230–239, 2002.

N. Tideman. The single transferable vote. *Journal of Economic Perspectives*, 9(1):27–38, 1995.

A. Ukkonen. Visualizing sets of partial rankings. In *Advances in Intelligent Data Analysis VII*, pages 240–251, 2007.

A. Ukkonen. *Algorithms for Finding Orders and Analyzing Sets of Chains*. PhD thesis, Helsinki University of Technology, 2008.

A. Ukkonen and H. Mannila. Finding outlying items in sets of partial rankings. In *Knowledge Discovery in Databases: PKDD 2007*, pages 265–276, 2007.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.