Publication I

Markus Ojala, Niko Vuokko, Aleksi Kallio, Niina Haiminen, and Heikki Mannila. 2009. Randomization methods for assessing data analysis results on real-valued matrices. Statistical Analysis and Data Mining, volume 2, number 4, pages 209-230.

Randomization Methods for Assessing Data Analysis Results on Real-Valued Matrices

Markus Ojala[1], Niko Vuokko[1], Aleksi Kallio[2], Niina Haiminen[3,4] and Heikki Mannila[1,3,*]

[1] *HIIT, Department of Information and Computer Science, Helsinki University of Technology, Finland*

[2] *CSC - IT Center for Science Ltd, Finland*

[3] *HIIT, Department of Computer Science, University of Helsinki, Finland*

[4] *Current address: IBM Watson Research Center*

Received 19 December 2008; revised 7 April 2009; accepted 8 April 2009
DOI:10.1002/sam.10042
Published online 24 September 2009 in Wiley InterScience (www.interscience.wiley.com).

Abstract: Randomization is an important technique for assessing the significance of data analysis results. Given an input dataset, a randomization method samples at random from some class of datasets that share certain characteristics with the original data. The measure of interest on the original data is then compared to the measure on the samples to assess its significance. For certain types of data, e.g., gene expression matrices, it is useful to be able to sample datasets that have the same row and column distributions of values as the original dataset. Testing whether the results of a data mining algorithm on such randomized datasets differ from the results on the true dataset tells us whether the results on the true data were an artifact of the row and column statistics, or due to some more interesting phenomena in the data. We study the problem of generating such randomized datasets. We describe methods based on local transformations and Metropolis sampling, and show that the methods are efficient and usable in practice. We evaluate the performance of the methods both on real and generated data. We also show how our methods can be applied to a real data analysis scenario on DNA microarray data. The results indicate that the methods work efficiently and are usable in significance testing of data mining results on real-valued matrices. © 2009 Wiley Periodicals, Inc. Statistical Analysis and Data Mining 2: 209–230, 2009

Keywords: significance testing; real-valued matrix; data mining; randomization tests; Markov chain Monte Carlo; DNA microarray

1. INTRODUCTION

Data mining research has produced many efficient algorithms for extracting knowledge from large masses of data. An important consideration is deciding whether the discovered patterns or models are significant in some sense. Traditional statistics have, since long, been considering the issue of significance testing, but it has been given less attention in the data mining community.

In statistics, the methods for significance testing are typically based either on analytical expressions or on randomization tests. We focus on randomization tests for assessing the significance of the discoveries; see Refs. [1–3] for excellent overall descriptions of randomization approaches in significance testing.

Given a structural measure, such as the clustering error for a given number of clusters or the maximum lift of an association rule, randomization approaches produce multiple random datasets according to a null hypothesis about the data. If the structural measure of the original data deviates significantly from the structural values of the random datasets, we can consider the original result as significant.

In this paper, we consider real-valued matrices where columns can be viewed to be of the same type, and rows, likewise. Gene expression data is a good example of such data: the rows correspond to genes and the columns correspond to samples, e.g., different tissues. The value of an element of the matrix indicates the level of activity of a gene in the sample. The rows have the same type, as do

Correspondence to: Heikki Mannila
(Heikki.Mannila@tkk.fi)

the columns. A dataset with columns for, say, weight and height of an individual would not satisfy our requirement.

We study randomization with respect to two null hypotheses. The first one is that the structure in the data is due to the row and column distributions of values. The second is that the structure is due to the row and column means and variances. These null hypotheses correspond to viewing the results of a data mining algorithm as interesting, if they are highly unlikely to be observed in a random dataset that has approximately the same row and column statistics. Thus, we can answer questions such as "Does the observed clustering or correlation structure convey any information in addition to the row and column statistics?"

The problem of generating matrices having the same row and column statistics as the input matrix is a generalization of the task considered in Refs. [4–6]. There the input is a 0–1 matrix, and the goal is to generate 0–1 matrices having the same row and column sums. This problem has been studied extensively in statistics, theoretical computer science, and various application areas [4,7–11], and it is computationally quite hard.

We describe simple algorithms for generating real-valued matrices that approximately share given row and column statistics. The algorithms are based on local transformations and Metropolis sampling. We evaluate the performance of the methods both on real and generated data, and show that the methods are usable in practice.

A preliminary version of this paper considered the case where only row and column means and variances are preserved [12]. Here we also study the more general problem of preserving row and column distributions. All the algorithms of the previous paper are revised, and more general versions of the methods are given. Additionally, we prove more theoretical properties of the methods and give extensive experimental results.

The rest of this paper is organized as follows. In Section 2, we give a brief overview of related work. In Section 3, we discuss applying randomization in significance testing, and give a simple example of randomizing real-valued matrices. In Section 4, we present local modification operations, describe a general Metropolis algorithm, and give a simple method based on discrete swaps for producing randomized datasets. We also give examples of different randomizations. In Section 5, we discuss the probability distribution of randomized matrices and introduce the difference measures for two different statistics. In Section 6, we describe the implementation of the methods. The algorithms are analyzed theoretically in Section 7. Our main experimental results are discussed in Section 8. In Section 9, we give some practical guidelines for using the methods in significance testing. Section 10 concludes the paper.

2. RELATED WORK

We are not aware of any work that would directly address the problem considered in this paper. Obviously, significance testing has received a large amount of attention. Excellent general sources on a variety of randomization approaches are [1–3].

Defining the significance of discovered patterns has attracted a lot of attention in data mining. The χ^2-test is used in Ref. [13] for significance testing of correlation rules, which are generalizations of association rules. Functional dependencies and logic are used for pruning out nonsignificant patterns algorithmically in Refs. [14,15]. Using inference to prune out nonsignificant correlations quickly was done in Ref. [16]. A more methodological view on pruning nonsignificant patterns using multiple hypotheses testing concepts can be found in Ref. [17]. Definitions and views on patterns other than frequent item sets or association rules can be found in Refs. [18–20].

Various null models have been studied in many application areas. In ecology, the use of null models in testing the significance of discoveries is quite widespread. For example, in the analysis of nestedness, there are several different null models whose properties differ somewhat, and which have been under careful study in recent years [21–23]. Null models for temporal trends in biological records are studied in Ref. [24] and in geographic range size evolution in Ref. [25].

Sampling from the space of contingency tables with fixed marginal sums has been widely studied [4,26,27]. A sequential importance sampling procedure for analyzing contingency tables is introduced in Ref. [7]. The asymptotics on the exact number of such tables is studied, e.g., in Ref. [28]. The algorithmic properties of some of the methods are discussed in Ref. [29]. A good survey on the topic is provided by Chen *et al.* [7]. The problem of generating random matrices with fixed margins has also been studied in many application areas, such as ecology [30] and biology [10], and analysis of complex networks [11].

3. APPLYING RANDOMIZATION IN SIGNIFICANCE TESTING

In this section we briefly describe the basic approach, give the definition of empirical p-values, and give an example of the usefulness of preserving row and column statistics when randomizing real-valued matrices. We also discuss how the approach is used to assess the significance of data mining results. See Refs. [1–3] for additional background.

3.1. Basic approach

Consider an $m \times n$ real-valued matrix A. Assume that some data mining task, such as clustering, is performed on A. Assume that the result of the data mining algorithm can be described by a single number $\mathcal{S}(A)$ which we call a *structural measure* of A. The structural measure can be, e.g., the clustering error of the matrix, the correlation between some specific columns, or the number of correlations above some threshold; any measure can be used, as long as it can be summarized by one number so that smaller (or larger) values mean stronger presence of structure.

To assess the significance of $\mathcal{S}(A)$, the idea is to generate randomized matrices $\hat{A}$ sharing some statistics with A, and to compare the original structural measure $\mathcal{S}(A)$ against the distribution of structural measures $\mathcal{S}(\hat{A})$. In this paper, we show how to generate randomized matrices independently and uniformly from the set of all matrices sharing approximately the distributions of values in rows and columns with the original matrix. Thus, the computational task we are addressing is the following.

TASK 1: *Given an $m \times n$ real-valued matrix A, generate a matrix $\hat{A}$ chosen independently and uniformly from the set of $m \times n$ real-valued matrices having approximately the same value distributions in rows and columns as A.*

We will also specify later what *approximately* means above. Additionally, we study a variant where only row and column sums and variances are preserved in randomization. That computational task is the following.

TASK 2: *Given an $m \times n$ real-valued matrix A, generate a matrix $\hat{A}$ chosen independently and uniformly from the set of $m \times n$ real-valued matrices having approximately the same row and column means and variances as A.*

3.2. Empirical p-Values

Let $\hat{\mathcal{A}} = \{\hat{A}_1, \ldots, \hat{A}_k\}$ be a set of randomized versions of the original matrix A. Then the one-tailed *empirical p-value* of the structural measure $\mathcal{S}(A)$, with the hypothesis of $\mathcal{S}(A)$ being small, is

$$\frac{\left|\left\{\hat{A} \in \hat{\mathcal{A}} \,|\, \mathcal{S}(\hat{A}) \leq \mathcal{S}(A)\right\}\right| + 1}{k+1}. \tag{1}$$

This captures the fraction of randomized matrices that have a smaller value of the structural measure than the original matrix. The one-tailed empirical p-value with the hypothesis of $\mathcal{S}(A)$ being large, and the two-tailed empirical p-value are defined similarly. If the p-value is small, we can say that the structural measure of the original matrix is significant and not due to the row and column statistics.

In practice, we will be generating the set $\hat{\mathcal{A}} = \{\hat{A}_1, \ldots, \hat{A}_k\}$ of randomized versions of A by using a Markov chain. In this approach, care has to be taken since the samples are not necessarily independent. We use the ideas from Refs. [2,3,31], where exchangeability of the samples $\hat{A}_i$ is guaranteed by first running the chain backward to some state $\hat{A}_0$ and then k times separately forward from state $\hat{A}_0$; see Section 4.5 for more details.

3.3. Example

Most existing randomization techniques for real-valued matrices are based on simply permuting the values in a single column (or row). To show why this is not necessarily enough, consider the two 10×5 real-valued matrices A and B shown in Figure 1. They share their first two columns, and the correlation between these columns x and y is high, 0.92. In matrix B the values on each row are tightly distributed around the mean of the row, whereas in matrix A the variance of each row is high. If the test of significance of correlation between columns x and y would consider only the first two columns, the results for the two matrices would be identical. However, it seems plausible that the high correlation between the first and second columns in matrix B is due to the general structure of the matrix, and not some interesting local structure involving the two columns, as might be the case with matrix A. More specifically, it seems that the correlation of x and y in B can be explained by the small variance of each row.

To test this observation, we generated randomized matrices: sets $\mathcal{A}$ and $\mathcal{B}$ each contain 999 independent random matrices having approximately the same value distributions in rows and columns as A and B. For the matrices in set $\mathcal{A}$ the smallest correlation between x and y is -0.85, the maximum 0.83, average -0.03 and standard deviation 0.30, while in set $\mathcal{B}$ corresponding values are 0.71, 0.99, 0.92, and 0.04, respectively. The empirical p-values of the correlation between x and y are 0.001 for matrix A and 0.52 for matrix B. Thus, we can conclude that the high correlation between the first and second columns in B is indeed due to the row and column distributions, while this is not the case for A. Similar results are obtained if only row and column means and variances are preserved in the randomization instead of value distributions.

The example illustrates that the structure of the entire matrix can have a strong effect on the significance of the results of very simple analysis methods.

3.4. Using the approach

In the previous example, we saw how the same value of a structural measure can have different significances

x	y			
.46	.36	.21	.68	.45
.44	.29	.64	.21	.04
.74	.87	.32	.84	.03
.04	.06	.96	.63	.31
.75	.66	.73	.13	.01
.85	.81	.41	.21	.38
.80	.98	.74	.61	.68
.70	.72	.27	.63	.09
.30	.37	.44	.37	.04
.57	.41	.93	.58	.61

Matrix A

x	y			
.46	.36	.56	.51	.53
.44	.29	.49	.52	.38
.74	.87	.90	.79	.80
.04	.06	.03	.11	.05
.75	.66	.68	.75	.71
.85	.81	.83	.81	.90
.80	.98	.88	.90	.81
.70	.72	.67	.79	.63
.30	.37	.37	.35	.43
.57	.41	.46	.44	.41

Matrix B

Fig. 1 Examples with two real-valued data matrices sharing the first two columns x and y having high correlation. The values on each row of the matrix B are close to each other, whereas in A the variance of each row is large. The high correlation between x and y is significant in A but not significant in B when tested using the methods introduced in this paper.

when the structure of the whole matrix is considered. The example could be described as follows. We start with a null model that requires matrices to have approximately the same marginal value distributions as the original data. Using different original matrices we obtain different null distributions. Using matrix A or B generates two different null distributions, so that the original correlation of 0.92 is an extreme value in the null distribution of $\mathcal{S}(A)$, but not in the null distribution of $\mathcal{S}(B)$.

The basic idea of the methods described in this paper is that they allow generating null distributions using the null model of the row and column value distributions. Thus we need not make assumptions on how data is distributed.

When randomization is used for significance testing, it is important to perform the exact same data mining procedure to the randomized matrices as to the original matrix to guarantee the correctness of the results. Take the previous example where we wanted to assess the significance of the correlation between the two columns. To obtain meaningful results, we need to use the same method to choose the two columns as we did on the original data. If they were just a pair of specific columns chosen without looking at the rest of the data, we should calculate the correlation between the same two columns in the randomized samples. Suppose, on the other hand, that the columns were selected because the correlation between them was the largest of all pairs of columns. Then we should also calculate the maximum pairwise correlation of columns in the randomized samples.

While assessing significance of local structures, such as pairwise correlations, we can also perform multiple tests. In the previous example we tested only a single correlation, but we could have tested correlations between all pairs of columns, for example. It would be erroneous to accept all p-values that are less or equal to a given threshold α and conclude that the null hypothesis was rejected at the significance level α. A good method for addressing multiple testing is to use, e.g., the Benjamini–Hochberg procedure that controls the false discovery rate (the proportion of false positives) [32].

4. ALGORITHMS

4.1. Introduction

In this section, we introduce two Markov chain Monte Carlo (MCMC) methods for sampling from the set of real-valued matrices with given row and column statistics. In general, MCMC methods are a class of algorithms for sampling from probability distributions. They are based on constructing a Markov chain that has the desired probability distribution as its stationary distribution. The state of the chain after a large number of steps is then used as a random sample from the desired distribution. For more information on MCMC methods, see, e.g., Ref. [3].

Both methods that we introduce output a randomized version $\hat{A}$ of the original $m \times n$ matrix A. Each method performs a random walk in a Markov chain, whose state space is the set of $m \times n$ matrices. Each step in the walk is a local modification of the current matrix. That is, the algorithms start from the original dataset A. Given a current dataset $\hat{A}_i$, the next step selects at random some local modification from the collection of allowed operations, and applies it to $\hat{A}_i$. If the change is accepted, this yields $\hat{A}_{i+1}$. Otherwise $\hat{A}_{i+1} = \hat{A}_i$.

The data is assumed to be scaled to the unit interval $[0, 1]$. To randomize a general matrix $A \in \mathbb{R}^{m \times n}$, we first scale it linearly into $[0, 1]$, randomize, and finally unscale it back to the original value range. In scaling, the original minimum value is replaced by zero and the maximum value by one. If there exist natural minimum and maximum values for the underlying phenomenon, they can be used instead. When studying a set of matrices produced by a similar phenomenon, the same scaling can be used, which ensures that the significance tests are not biased.

We also discuss how to apply the algorithms for producing an exchangeable set of samples for calculating a valid p-value, and give some visual examples of randomizations with different approaches. The notation A_{ij} refers to the element at row i and column j in a matrix A.

The implementations of the algorithms are available at http://www.cis.hut.fi/mrojala/randomization/.

4.2. General Metropolis Method for Randomizing a Matrix

We describe a general Metropolis algorithm [33,34] for randomizing a real-valued matrix while preserving some

statistics. The idea is to generate samples $\hat{A}$ from the probability distribution

$$\Pr(\hat{A}|A) = c \exp\{-wE(\hat{A}, A)\}, \quad (2)$$

where A is the original matrix, $w > 0$ is a constant, c is a normalizing constant and $E(\hat{A}, A)$ is any difference measure of the sample $\hat{A}$. That is, we wish to sample matrices so that the matrices $\hat{A}$ for which $E(\hat{A}, A)$ is small have a high probability of being generated. For example, $E(\hat{A}, A)$ can measure the total difference in row and column value distributions between $\hat{A}$ and A. The constant w controls how steep the distribution is. The probability distribution 2 is discussed in detail in Section 5.

The Metropolis method is a general technique for obtaining samples from a probability distribution π [33]. Let S denote the set on which π is defined, and let $Q(x, y)$ be a symmetric proposal distribution on S, i.e., $\sum_y Q(x, y) = 1$ and $Q(x, y) = Q(y, x)$ for all $x, y \in S$. Then the Metropolis method samples y from state x with probability $Q(x, y)$ and moves to state y with probability

$$\min(1, \pi(y)/\pi(x)).$$

A direct implementation of the Metropolis approach for generating samples $\hat{A}$ from the probability distribution 2 is presented in Algorithm 1.

Algorithm 1 GeneralMetropolis(A, A_0, I, w)

Input: Original matrix A, starting matrix A_0, number of attempts I, parameter $w > 0$
Output: Randomized matrix $\hat{A}$

```
1: Â ← A_0
2: for i ← 1, I do
3:    A' ← LocalModification(Â)
4:    if ∀i, j : A'_ij ∈ [0, 1] then
5:       u ← Uniform(0, 1)
6:       if u < exp{−w[E(A', A) − E(Â, A)]} then
7:             Â ← A'
8:       end if
9:    end if
10: end for
11: return Â
```

We use local modifications to obtain the proposals from the distribution $Q(x, y)$; the function LocalModification(A) returns a possible variation of A. Symmetry of the proposal distribution means that the probability of the events $B =$ LocalModification(A) and $A =$ LocalModification(B) must be equal. Additionally, we require that the values after a local modification are in the original range, which is assumed to be $[0, 1]$. In Section 4.3, we introduce some local modifications and discuss their properties. Note, however, that the method *GeneralMetropolis* can be used with any local modification operation.

Table 1. Abbreviations and short descriptions of the local modifications introduced.

Name	Description
`Change`	Replace one element uniformly from [0, 1]
`Resample`	Replace one element by resampling from A
`Add`	Add a uniform number from $[-s, s]$
`Rotate`	Rotate four intersection elements
`Mask`	Add a mask to four intersection elements

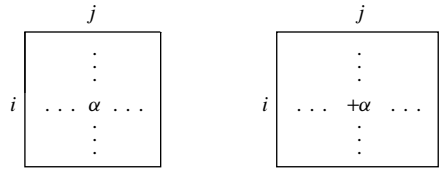
4.3. Local Modifications

We introduce several local modification operations. Some only shuffle the values in A, while some assign new values to elements of the matrix on the basis of some distribution. They change either one or four elements of a matrix in each step. For the needs of our methods, we additionally require that the probability of making a local modification that undoes the effect of a local modification M is the same as the probability of M. Table 1 summarizes the local modifications.

The local modifications are ad-hoc in nature but they have a large impact on the speed and quality of randomization. A good local modification should be fast in performance, change the matrix as much as possible while introducing only a small difference in the row and column statistics. These objectives are contradictory. Each local modification we introduce is good in different aspects. The local modifications are compared to each other more in the experiments.

4.3.1. *Replacing a Single Value:* `Change`, `Resample`, `Add`

We introduce three simple local modifications that are based on replacing a single element with another element. The element to be changed is chosen uniformly at random from among all elements of $\hat{A}$. The simplest local modification, `Change`, replaces the original value with a random value drawn uniformly from $[0, 1]$. The next local modification, `Resample`, replaces the element with one of the original values of A. This guarantees that the original value distribution of the whole matrix is preserved approximately. The last simple local modification, `Add`, adds a randomly chosen value α from $[-s, s]$ to the current value. The parameter s controls the scale of the transformation. In the experiments, we will use a parameter value $s = 0.1$. Also, other distributions than uniform could be used for choosing the value to add. These three local modifications



(a) Replace value (b) Add value.

Fig. 2 Three simple local modifications based on value replacement (`Change` and `Resample`) and value addition (`Add`). Each change only one element at a time.

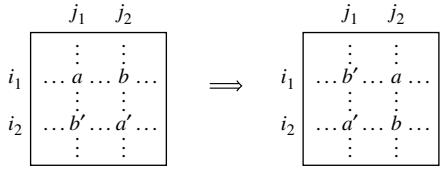


Fig. 3 An example of local modification `Rotate`. The four elements shown are rotated and the rest of the matrix is kept fixed. If $a = a'$ and $b = b'$, then the row and column statistics do not change.

are depicted in Figure 2. Note that in each case the symmetry of the proposal distribution is obtained.

4.3.2. *Swap Rotation:* `Rotate`

The next local modification, `Rotate`, is a generalization of binary swaps. The idea of swapping matrix elements as a randomization technique has a long history [4]. Here we use a concept of swap rotations as shown in Figure 3, which degenerates to conventional swaps in the case of binary data. At each step, we randomly choose from the current matrix four elements a, b, a', and b', located at the intersections of two rows i_1 and i_2 and two columns j_1 and j_2. A new matrix is produced by rotating those four elements clockwise, while keeping the other elements unchanged. Again, it is clear that the proposal distribution is symmetric.

The smaller the difference between (a, b) and (a', b'), the smaller the change in the row and column statistics will be. If $a = a'$ and $b = b'$, the row and column statistics do not change at all, corresponding to binary swaps. In Section 4.4 we will introduce a method called *SwapDiscretized* which utilizes this property.

4.3.3. *Addition Mask:* `Mask`

Our next modification, `Mask`, preserves the row and column sums exactly. As with swap rotation, a new matrix is created from the current one by selecting rows i_1, i_2 and columns j_1, j_2 at random, and adding the mask presented in Figure 4 to the four intersection elements. The value α

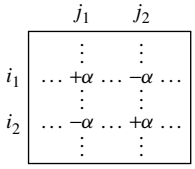


Fig. 4 An example of local modification `Mask`. The addition mask preserves the row and column sums.

is drawn uniformly at random from the range $[-s, s]$. In the experiments, we use parameter value $s = 0.1$.

4.4. Discrete Swaps

We will see that maintaining the difference measure $E(\hat{A}, A)$ is troublesome. Therefore, we also consider a variant based on discretization that avoids keeping track of the difference in the distributions. Denote by $\text{Class}(x, N)$ the function that discretizes $x \in [0, 1]$ into a value in $1, \ldots, N$. We use the discretization where the range $[0, 1]$ is divided into N intervals of equal length, thus, $\text{Class}(x, N) = \min(\lfloor Nx \rfloor, N - 1)$.

The method *SwapDiscretized* is based on rotations (recall Figure 3). At each step, the method samples indices i_1, i_2 of rows and j_1, j_2 of columns. Given the number C of classes for values in columns and the number R of classes for values in rows, the rotation is accepted if

$$\text{Class}(\hat{A}_{i_1 j_1}, C) = \text{Class}(\hat{A}_{i_2 j_2}, C) \quad \text{and}$$
$$\text{Class}(\hat{A}_{i_1 j_2}, R) = \text{Class}(\hat{A}_{i_2 j_1}, R).$$

This guarantees that the distributions of the discretized values in the rows and columns do not change.

The restrictions that $\hat{A}_{i_1 j_1}$ and $\hat{A}_{i_2 j_2}$ belong to the same column class as well as $\hat{A}_{i_1 j_2}$ and $\hat{A}_{i_2 j_1}$ to the same row class can decrease the acceptance rate dramatically. However, we can select $\hat{A}_{i_1 j_1}$ and $\hat{A}_{i_2 j_2}$ belonging to the same column class in constant time. This is possible if we keep track of where the elements in each class are located. First, we randomly select an element $\hat{A}_{i_1 j_1}$ from the matrix, and after that, we randomly select $\hat{A}_{i_2 j_2}$ that belongs to the same class as $\hat{A}_{i_1 j_1}$. The pseudocode of this approach is presented in Algorithm 2.

Note that at each step the next state can be the current state, i.e., there are self-loops in the state space.

4.5. Obtaining Exchangeable Samples

Subsequent samples produced by the Metropolis algorithm are dependent, unless the number of steps taken between the samples is at least the mixing time. It is very hard to estimate this quantity in any application. We use

Algorithm 2 SwapDiscretized(A, I, R, C)

Input: Matrix A, number of attempts I, number of classes in rows R and columns C
Output: Randomized matrix $\hat{A}$
1: $\hat{A} \leftarrow A$
2: **for** $i \leftarrow 1, I$ **do**
3: Pick i_1 and j_1 randomly
4: Pick i_2 and j_2 randomly with Class($\hat{A}_{i_1 j_1}, C$) = Class($\hat{A}_{i_2 j_2}, C$)
5: **if** $i_1 \neq i_2$ and $j_1 \neq j_2$ and Class($\hat{A}_{i_1 j_2}, R$) = Class($\hat{A}_{i_2 j_1}, R$) **then**
6: $\hat{A} \leftarrow$ Rotate($\hat{A}, i_1, i_2, j_1, j_2$)
7: **end if**
8: **end for**
9: **return** $\hat{A}$

the ingenious technique of Besag and Clifford [2] to obtain a set of exchangeable samples.

The technique of Besag and Clifford consists of first running the chain I steps backward. That is, given the original dataset A, we obtain a set $\hat{A}_0$ such that there is a path of length I from $\hat{A}_0$ to A. Then for the desired number k of samples, we start for each $i = 1, \ldots, k$ from $\hat{A}_0$ and run the chain I steps forward, obtaining samples $\hat{A}_i$. Then $\{A, \hat{A}_1, \ldots, \hat{A}_k\}$ forms an exchangeable set of samples.

That is, if the null hypothesis is true for the original dataset, A, then the samples $A, \hat{A}_1, \ldots, \hat{A}_k$ have an underlying joint distribution that is exchangeable. Thus, the rank of $\mathcal{S}(A)$ among values $\{\mathcal{S}(A), \mathcal{S}(\hat{A}_1), \ldots, \mathcal{S}(\hat{A}_k)\}$ is uniform, implying the validity of the empirical p-value regardless of the irreducibility and convergence of the chain. The result may just be more conservative.

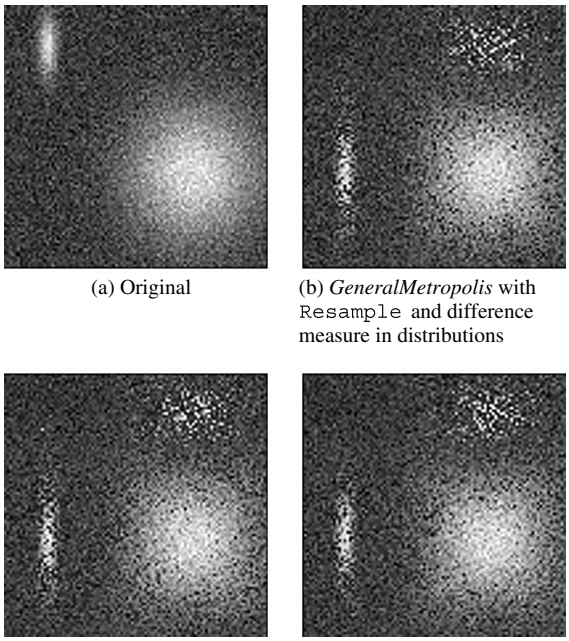
In our case, running the chain backward turns out to be very easy, as the chain is time-reversible. Thus, running the chain backward is the same as running it forward. The method is given in Algorithm 3. The same method can also be used with *SwapDiscretized*.

4.6. Examples of Randomizations

Next, we give some visual examples of the results produced by different randomization approaches. In Figure 5

Algorithm 3 Besag-Clifford(A, k, I, w)

Input: Matrix A, number of samples k, number of attempts I, parameter w
Output: Exchangeable set of randomized samples $\{\hat{A}_1, \ldots, \hat{A}_k\}$
1: $\hat{A}_0 \leftarrow$ GeneralMetropolis(A, A, I, w)
2: **for** $i \leftarrow 1, k$ **do**
3: $\hat{A}_i \leftarrow$ GeneralMetropolis($A, \hat{A}_0, I, w$)
4: **end for**
5: **return** $\{\hat{A}_1, \ldots, \hat{A}_k\}$



(a) Original
(b) *GeneralMetropolis* with `Resample` and difference measure in distributions
(c) *General Metropolis* with `Mask` and difference measure in means and variances
(d) *SwapDiscretized*

Fig. 5 Original data and results of randomization of the original data with three different methods. Black corresponds to zero and white to one. The small top left artifact in the original matrix has disappeared in randomizations, which have produced "shadows" of the artifact to the top right and bottom left regions. The randomizations resemble each other although different randomization approaches were used.

randomizations of a 100×100 matrix resembling a hilly surface are shown. The results come from applying our algorithms with the parameters described in Section 8.2.

The three different randomization methods have produced similar results. In the randomized matrices, the massive bottom right hill remains, but the smaller top left structure disappears. The existence of the small hill on the top left in the original data results in shadow shapes emerging in the top right and bottom left regions in randomization.

5. PROBABILITY DISTRIBUTION FOR RANDOMIZED MATRICES

Let A be the original $m \times n$ real-valued matrix whose row and column statistics we wish to maintain. We discuss the probability distribution $\Pr(\hat{A}|A)$ from which the randomized $m \times n$ real-valued matrices $\hat{A}$ are drawn. The probability distribution is defined by using a general difference measure between two sets of real values, such as the sets of values in a specific row of the original and a randomized matrix. We introduce two difference measures corresponding to Tasks 1 and 2 in Section 3.1. The first

one measures the difference in value distributions and the second one the difference in means and variances.

5.1. Decomposition of Probability Distribution

Let r_i be the set of values in ith row of A and c_j the set of values in jth column of A. Let $\hat{r}_i$ and $\hat{c}_j$ be the corresponding sets in a randomized matrix $\hat{A}$. Note that r_i and c_j are not independent as they both contain the value A_{ij}. However, we use an approximation of $\Pr(\hat{A}|A)$ that assumes the independence of rows and columns. Thus we write

$$\Pr(\hat{A}|A) = \prod_{i=1}^{m} \Pr(\hat{r}_i|r_i) \cdot \prod_{j=1}^{n} \Pr(\hat{c}_j|c_j). \tag{3}$$

Let s be an original set of values corresponding to r_i or c_j, and let $\hat{s}$ be another set of values whose probability we would like to estimate given the original set s. Let $E(\hat{s}, s)$ be any difference measure between the two sets s and $\hat{s}$. We define the probability of $\hat{s}$ as

$$\Pr(\hat{s}|s) = c \exp\{-wE(\hat{s}, s)\},$$

where w is a scaling constant and c is a normalizing constant. Denoting by $E(\hat{A}, A)$ the total difference in the maintained statistics in the rows and columns between a randomized matrix $\hat{A}$ and the original matrix A,

$$E(\hat{A}, A) = \sum_{i=1}^{m} E(\hat{r}_i, r_i) + \sum_{j=1}^{n} E(\hat{c}_j, c_j), \tag{4}$$

we have that Equation 3 implies Equation 2 given in Section 4,

$$\Pr(\hat{A}|A) = c \exp\{-wE(\hat{A}, A)\}.$$

5.2. Difference in Distributions

There exist various distance measures between two probability distribution functions (pdf), such as Kullback-Leibler divergence, direct L_1 distance between two probability or cumulative distribution functions (cdf) [35,36]. We will use the L_1 distance between the cumulative distribution functions as it works well when only a sample set of a distribution is available.

Let D_s be the unnormalized empirical cdf of the set s defined by

$$D_s(x) = |\{y \in s \mid y \leq x\}|.$$

The normalized version of the empirical cdf is obtained by dividing $D_s(x)$ by $|s|$. Let $D_{\hat{s}}(x)$ be the corresponding unnormalized empirical cumulative distribution function of the set $\hat{s}$ with $|\hat{s}| = |s|$. We will use the L_1 norm between two unnormalized empirical cdfs as the difference measure in distributions. Thus, we define

$$E_D(\hat{s}, s) = \|D_s - D_{\hat{s}}\|_{L_1} = \int_{-\infty}^{\infty} |D_s(x) - D_{\hat{s}}(x)| \, \mathrm{d}x. \tag{5}$$

By changing the order of integration and summation the unnormalized cdf difference 5 can be written in an easier form

$$E_D(\hat{s}, s) = \sum_{i=1}^{|s|} |s_i - \hat{s}_i|, \tag{6}$$

where s_i and $\hat{s}_i$ are the values in s and $\hat{s}$, respectively, in increasing order.

We use the unnormalized version of the cdf: it provides a good implicit weighting between the difference in rows and in columns even if their sizes differ from each other. The change in the cdf difference after modifying a single element is approximately independent of the size of the vector. Thus, the larger a row or column is the smaller the scaled cdf difference will be.

5.3. Difference in Means and Variances

Measuring the difference in means and variances is easier than measuring the difference in distributions. Let μ and σ^2 be the mean and the variance of s. Let $\hat{\mu}$ and $\hat{\sigma}^2$ be the mean and the variance of $\hat{s}$, respectively. We define the difference in means and variances as

$$E_{MV}(\hat{s}, s) = |s| \cdot (|\mu - \hat{\mu}| + |\sigma - \hat{\sigma}|), \tag{7}$$

where we scale by $|s|$ for the same reason as we used the unnormalized empirical cdf in the previous subsection. The difference can be calculated when the sums and the sums of squares of s and $\hat{s}$ are known.

Note that the difference measures E_D and E_{MV} are not scale invariant, i.e., multiplying A and $\hat{A}$ both by the same constant changes the value of $E(\hat{A}, A)$. Recall, however, that the values A_{ij} are assumed to be in the interval $[0, 1]$. This fixes the problem partly. The final touch is done by tuning the scaling constant $w > 0$ separately for each data.

6. UPDATING THE DIFFERENCE MEASURES AFTER A LOCAL MODIFICATION

The time complexity of the *GeneralMetropolis* algorithm is dominated by the calculation of the change $E(A', A) -$

Table 2. Three difference measures and short descriptions.

Name	Description
GM-MeanStd	Difference in means and variances 7
GM-Cdf	Difference in cdfs 6
GM-CdfHist	Histogram approximation of *GM-Cdf*

$E(\hat{A}, A)$ on line 6. In this section, we discuss how the change in the difference measures can be calculated efficiently after a local modification. Recall that the total difference $E(\hat{A}, A)$ of a matrix $\hat{A}$ defined by Equation 4 is a sum of differences of rows and columns. Thus, after a local modification the change in the differences in the maintained statistics depends only on the rows and columns where the values were modified.

Since all our local modifications change either one or four elements of the matrix at a time, it is enough to consider how we can update the difference after changing one element. The four elements case can be obtained by performing four single changes. With all difference measures we keep track of the current difference on each row and column in appropriate data structures. Changing a value in a matrix affects only the corresponding row and column data structures. Thus, it is enough to consider what kind of data structure is needed for a single set $\hat{s}$ of real-values and how it is updated to a new set $s' = \hat{s} \cup \{y\} \setminus \{x\}$.

Updating the difference when only row and column means and variances are preserved is easy. Updating the difference in cdfs is hard. Thus, we introduce a histogram approximation of cdfs that approximately measures the difference in cdfs and is easier to update. Table 2 summarizes the matrix difference measures.

6.1. Means and Variances: *GM-MeanStd*

Updating the difference in means and variances after a local modification is simple. For each row and column we keep track of the sum and the sum of squares of the elements. We store also the sum and the sum of squares of the original matrix. With these values the difference $E(\hat{s}, s)$ defined in Equation 7 can be calculated in constant time. Updating the sum and the sum of squares after changing a value takes a constant time. Thus, calculating the change in differences after a local modification takes constant time.

6.2. Cumulative Distribution Functions: *GM-Cdf*

Keeping track of the difference in cdfs as defined in Equation 5 is computationally demanding. For each row and column we store the original values as well as the current values in arrays in increasing order. Initializing these takes $O(mn \log(mn))$ time for a matrix with m rows and n columns. Calculating the L_1 cdf difference directly between two distributions D and $\hat{D}$ takes time $O(|s|)$, when the values are given in increasing order.

Updating the distribution difference 6 when a single value $x \in \hat{s}$ is changed to a new value y may take, in worst case, time $O(|s|)$. The update is done by moving the new value, y, into the correct position to make the array corresponding to s' sorted in increasing order. After that we only need to calculate the part in the summation in Equation 6 where the ordering has changed. Thus, only the values $z \in \hat{s}$ between x and y have to be considered. Finding the place of the original value x in $\hat{s}$ takes time $O(\log(|s|))$ and moving the new value y to the correct place and calculating the change in differences takes time $O(l)$ where l is the number of elements between x and y.

The update is easy if the cdf curves of s and $\hat{s}$ do not intersect between x and y. Then the change in differences is simply $\pm|x - y|$. However, when the D and $\hat{D}$ intersect frequently, it is easiest to iterate over all elements in s and $\hat{s}$ between x and y. Thus, in our case, we have to cope with the update time $O(l + \log(|s|))$. However, by choosing an appropriate local modification the size of l can be made small on average.

6.3. Histogram Approximation of CDFs: *GM-CdfHist*

Since computing the difference in cdfs requires time linear in the size of the data, we next introduce a new difference measure that measures the cdf difference approximately but is faster to calculate. We approximate the distributions by histograms, with the range $[0, 1]$ divided uniformly into N bins. Thus, in practice, we distinguish only N different values. We use a histogram approximation by Cha *et al.* [36].

For each row and column we use N numbers to store the differences of the cumulative distributions for each of the N bins. Initializing these takes time $O(mn + N(m + n))$. When a value is changed, we have to change the cumulative values in l bins, where l is the number of bins whose corresponding value is between the old and new value, inclusive. This takes time $O(l)$. In the exact method l was the number of elements between the old and new values. Thus, the approximate method can yield significant speedup when n or m is large compared to the number of bins N. Additionally, we do not have to find the correct position of the old value by binary search as it can be obtained directly.

7. ANALYSIS OF THE METHODS

In this section, we discuss some of the properties of the introduced randomization methods. The methods are instantiations of MCMC methods for sampling from the

desired probability distribution. We study a few general properties of MCMC methods which are introduced here shortly: A Markov chain is said to be irreducible if it is possible to get to any state from any other state by using the corresponding local modification. The stationary distribution of the Markov chain is the distribution where the chain converges. The mixing time of a Markov chain is the number of steps needed for the convergence to the stationary distribution. We study these properties to make certain that the randomization methods produce samples with good quality in reasonable time.

7.1. Irreducibility of the Markov Chains

First, we give results on the irreducibility of the Markov chains of *SwapDiscretized* and *GeneralMetropolis* with different local modifications. The Markov chain of *GeneralMetropolis* is irreducible when the update operation is one of `Change`, `Resample`, `Add`, `Rotate` or `Mask`. The Markov chain of *SwapDiscretized* is, in general, reducible, i.e., not all states are reachable from others. However, as Besag *et al.* have stated, irreducibility of the chain is not essential for the validity of the p-value [3].

The following two theorems give the irreducibility results for local modifications with single value replacements. The proofs are trivial.

THEOREM 1: Given a real-valued matrix $A \in [0, 1]^{m\times n}$ all matrices $\hat{A} \in [0, 1]^{m\times n}$ are reachable from A by using the local modification `Change` or `Add`.

THEOREM 2: Given a real-valued matrix $A \in [0, 1]^{m\times n}$ all matrices $\hat{A} \in [0, 1]^{m\times n}$ containing only the values of A (possibly multiple times) are reachable from A by using the local modification `Resample`.

The irreducibility of the Markov chains with the other local modifications that change four values at a time are less trivial. However, the following two theorems cover the irreducibility for them.

THEOREM 3: Given a real-valued matrix $A \in \mathbb{R}^{m\times n}$, $m \geq 3$, $n \geq 2$ or $m \geq 2$, $n \geq 3$, all matrices $\hat{A} \in \mathbb{R}^{m\times n}$ that contain the values of A permuted randomly are reachable from A by using the local modification `Rotate`.

PROOF 1: Consider the case $m \geq 2$ and $n \geq 3$. By brute force it can be shown that all permutations of a 2×3 matrix can be transformed to each other by using swap rotations. We can use this result to fix one element at a time of a $m \times n$ matrix until the whole matrix is in correct order. □

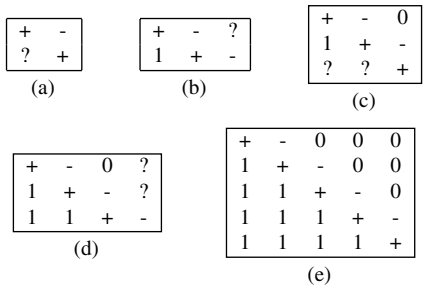


Fig. 6 Steps of the proof of Theorem 4. Plus means that $A_{ij} > \hat{A}_{ij}$ and minus that $A_{ij} < \hat{A}_{ij}$. When the process has stopped, the last row or column contradicts the sum assumption.

THEOREM 4: Given a real-valued matrix $A \in [0, 1]^{m\times n}$, all matrices $\hat{A} \in [0, 1]^{m\times n}$ whose row and column sums equal the corresponding values of A are reachable from A via $[0, 1]^{m\times n}$ matrices by using the local modification `Mask`.

PROOF 2: The hard part of the proof is the restriction to $[0, 1]^{m\times n}$ matrices. Otherwise, we could just fix one element at a time. We prove that we can transform A to any $\hat{A} \in [0, 1]^{m\times n}$ whose row and column sums equal to the corresponding values of A by using only addition masks that maintain the values within the range [0, 1] and satisfy one of the following two conditions:

1. the difference $\sum_{i,j} |\hat{A}_{ij} - A_{ij}|$ decreases
2. the number of zeros and ones in A decreases while the difference stays constant

If the difference $\sum_{i,j} |\hat{A}_{ij} - A_{ij}|$ is zero then $\hat{A} = A$ and we have been able to transform A to $\hat{A}$. Thus, suppose that the process gets stuck in a local minimum where the difference is nonzero and we cannot apply any addition mask that would satisfy one of the two conditions. We show that this leads to a contradiction.

As the difference is nonzero there is an element, say, $A_{1,1}$ that is greater than $\hat{A}_{1,1}$. As the sum of each row is fixed, there is an element on the first row, say, $A_{1,2}$ that is smaller than $\hat{A}_{1,2}$. Similarly, there is an element on the second column, say, $A_{2,2}$, that is greater than $\hat{A}_{2,2}$. See Figure 6(a). By using the addition mask on $\{A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}\}$ we could decrease the difference if $A_{2,1} < 1$. That would contradict our assumption. Thus, $A_{2,1} = 1$. Now $A_{2,1} \geq \hat{A}_{2,1}$ and $A_{2,2} > \hat{A}_{2,2}$, and hence, there is an element on the second row, say, $A_{2,3}$ that is smaller than $\hat{A}_{2,3}$ (see Figure 6(b)). Similarly, we get the next step presented in Figure 6(c).

Now we need also the second rule to decide what $A_{3,1}$ has to be. Consider the quartet $\{A_{1,1}, A_{1,3}, A_{3,1}, A_{3,3}\} = \{+, 0, ?, +\}$. The element $A_{3,1}$ has to be 1 as otherwise we

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \not\rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

Fig. 7 A counterexample of irreducibility of *SwapDiscretized*. The two matrices have the same row and column statistics, but they cannot be transformed to each other by using swap rotation, since there does not exist any rotatable quartet.

could decrease the number of zeros and ones in A without increasing the difference. Continuing this process by using similar kind of reasoning we get a matrix resembling the one presented in Figure 6(e). As the matrix has a finite size, the process stops at some step. If $n \leq m$ then the sum of row n will be larger than supposed, which is a contradiction. If $m < n$, then the sum of column m will be smaller than supposed, again a contradiction. Thus, we can always perform an addition mask that satisfies one of the two conditions if the difference is nonzero. Hence, the Markov chain is irreducible. □

The Markov chain of *SwapDiscretized* is reducible in general, as is easily seen. Consider the example shown in Figure 7 with $R = C = 3$. The matrices have the same number of entries of each type per row and column, but neither of them contain four elements that could be rotated. Thus, they cannot be transformed to each other. The counterexample can be generalized directly to all matrices with an odd number of rows or columns.

7.2. Stationary Distribution

The actual probability distribution from which *GeneralMetropolis* samples differs slightly from the distribution 2 introduced earlier. The final distribution is

$$\Pr(\hat{A}|A) = \begin{cases} c \exp\{-wE(\hat{A}, A)\}, & \hat{A} \in S, \\ 0, & \hat{A} \notin S, \end{cases} \quad (8)$$

where S is the set of all matrices $\hat{A} \in [0, 1]^{m \times n}$ that are reachable from A via $[0, 1]^{m \times n}$ matrices by using the given local modification pattern.

Next, we consider the stationary distribution of *SwapDiscretized*.

THEOREM 5: The stationary distribution of *SwapDiscretized* is uniform among all matrices that are reachable from the original matrix and have the original discretized row and column distributions.

PROOF 3: The stationary distribution of a reversible chain is proportional to the degree at each state in the underlying transition graph [3]. First we note that the chain of *SwapDiscretized* is reversible, as the probability of making a swap rotation that undoes a rotation is the same as the probability of making that rotation initially. The number of neighbors of each state is constant since we allow so called self-loops, i.e., the chain can stay in the same state. Thus, the stationary distribution of *SwapDiscretized* is uniform. □

7.3. Acceptance Probability of a Local Modification

Next, we analyze the probability of accepting a local modification. With *GeneralMetropolis* the parameter w fixes the amount of difference we allow in the maintained statistics. Thus, choosing the value of w involves making a compromise between efficiency of mixing and the difference induced in the row and column statistics: increasing w decreases the chance of accepting transitions that induce additional difference in the maintained statistics. The acceptance rate depends on the local modification as well as the difference measure used. In the experiments, we study the effects of these empirically.

However, for *SwapDiscretized* we can give a simple lower bound for the acceptance rate. The number of row classes R and column classes C involves making a compromise between efficiency of mixing, and the difference induced in the row and column statistics.

THEOREM 6: The acceptance probability of a swap in *SwapDiscretized* is approximately at least $1/R$ if the locations of class labels in matrix A are randomly distributed.

PROOF 4: Let n_l be the number of elements with row class label l, giving $\sum_{l=1}^{R} n_l = mn$. If $A_{i_1 j_2}$ has label l then the probability of accepting a swap in line 5 is $\frac{n_l - 1}{mn - 1} \approx \frac{n_l}{mn}$ since the locations of labels are assumed to be randomly distributed. Using Chebyshev's sum inequality or Cauchy-Schwarz inequality we get the result

$$\begin{aligned} \Pr(\text{Swap is accepted}) &= \sum_{l=1}^{R} \left(\frac{n_l}{mn}\right)^2 \\ &\geq R \left(\frac{\sum_{l=1}^{R} \frac{n_l}{mn}}{R}\right)^2 = \frac{1}{R}. \end{aligned} \quad (9)$$

□

Theorem 6 gives an interesting way to optimize *SwapDiscretized*. Since the acceptance rate depends only on the number of row classes R and not on the number of column classes C in Algorithm 2, we can exploit this asymmetry. If $R \leq C$ we can use the pseudocode presented in Algorithm 2. If, however, $C < R$ we can apply the algorithm to the transposed matrix A^T, and return the transpose of such randomized matrix. Effectively, this just changes

the acceptance rate from $1/R$ to $1/C$ and speeds up the randomization. We will use this optimization in the experiments giving the approximate lower bound $1/\min(R, C)$ for the acceptance rate.

7.4. Mixing Time

Finding the mixing time of an MCMC method is a theoretically hard issue. In the experiments, we will use some simple diagnostics to assess the convergence. Nevertheless, the following discussion gives some idea of the mixing times of our methods. We stress, however, that it is just a simplified reasoning, not a proof of the mixing time.

From the well-known coupon collector's problem we know that the expected number of attempts to collect each of n different coupons is $nH_n \approx n \ln n$ where H_n is the nth harmonic number. Our MCMC methods are based on swapping items, and thus, resemble this coupon-collector's problem. We can roughly assume that the chain has mixed when each element has been swapped at least once, because then each element can be assumed to be in a random position. Let the acceptance rate of the local modifications of the method be ρ. Then the number of attempts needed is approximately $mn \ln(mn)/\rho$.

From Theorem 6 and the discussion following it, we know that the acceptance rate of *SwapDiscretized* is approximately at least $\rho > 1/\min(R, C)$. Thus for the method *SwapDiscretized*, the number of attempts needed is approximately $\frac{1}{4}mn \ln(mn) \min(R, C)$, where the coefficient $\frac{1}{4}$ follows from the fact that four elements are touched at each swap. For example, if the matrix size is $m \times n = 1000 \times 100$ and we use $R = 20$, $C = 60$ classes, the approximation gives $58mn$ attempts.

7.5. Randomness of a Randomized Matrix

We introduce here a distance measure that can be used to qualify the difference between the original and a randomized matrix. We use it in the experiments to evaluate the randomness of the resulting matrices.

Let A be the original $m \times n$ real-valued matrix and $\hat{A}$ a randomized version of A. We define the *normalized root mean square distance* between A and $\hat{A}$ as

$$d(\hat{A}, A) = \frac{1}{\sigma_A} \sqrt{\frac{\sum_{i,j} |A_{ij} - \hat{A}_{ij}|^2}{mn}}, \tag{10}$$

where σ_A is the standard deviation of the values in A. Note that $d(\hat{A}, A)$ is just the scaled Frobenius distance between A and $\hat{A}$. For random permutations we can calculate the expected value of $d(\hat{A}, A)$, which can be used as a simple upper bound of randomness for the case where the row and column statistics are preserved. In a permutation of a matrix, the locations of the elements in the matrix are shuffled.

LEMMA 7: Let X be any dataset of size n, and let $P(X)$ denote a permutation of X selected uniformly at random. Then we have $\mathrm{E}[X \cdot P(X)] = \mathrm{E}[X]^2$.

PROOF 5: Let S_n be the symmetric group of order n, i.e., S_n contains all the permutations of n elements. Then

$$\begin{aligned}\mathrm{E}[X \cdot P(X)] &= \frac{1}{n!} \sum_{\pi \in S_n} \frac{1}{n} \sum_{k=1}^{n} X_k X_{\pi(k)} \\ &= \frac{1}{n \cdot n!} \sum_{k=1}^{n} X_k \sum_{\pi \in S_n} X_{\pi(k)} \\ &= \frac{1}{n^2} \sum_{k=1}^{n} X_k \sum_{i=1}^{n} X_i = \mathrm{E}[X]^2.\end{aligned}$$

□

THEOREM 8: The expected value of the mean square distance between a matrix A and its permutations equals $2\,\mathrm{Var}(A)$.

PROOF 6:

$$\begin{aligned}\mathrm{E}[(A - P(A))^2] &= \mathrm{E}[A^2] - 2\,\mathrm{E}[A \cdot P(A)] + \mathrm{E}[P(A)^2] \\ &= 2\,\mathrm{E}[A^2] - 2\,\mathrm{E}[A]^2 = 2\,\mathrm{Var}(A).\end{aligned}$$

□

Therefore, if $\hat{A}$ was a random permutation of A, we would expect that $d(\hat{A}, A) \approx \sqrt{2}$. However, as we are preserving row and column statistics, we are also restricting the randomness. Thus in practice, the distance $d(\hat{A}, A)$ of a good randomized matrix $\hat{A}$ may have any value between zero and $\sqrt{2}$ depending on the original matrix. We discuss this effect in the experiments.

8. EXPERIMENTAL RESULTS

In this section, we give results from experiments on synthetic and real datasets. We introduce a procedure for selecting the parameter values, and discuss the convergence, performance, and the difference in the maintained statistics between the original and randomized matrices. We apply the randomization methods in assessing the significance of three different structural measures: clustering error, maximum correlation between rows, and variance explained by the main principal components. We further compare our

methods to simple permutations methods, where either the entire matrix is permuted or where only the values in rows or columns are permuted. Finally, we demonstrate how to apply our methods in DNA microarray data analysis, as a case study of an existing application.

8.1. The Datasets

We used five types of artificial data in our experiments. The first dataset RANDOM contains 100 rows and 100 columns, with each entry independently generated from the normal distribution with zero mean and unit variance. The second dataset SUM is a 100×100 matrix where values r_i and c_j are drawn for each row and column from $\mathcal{N}(0, 1)$, respectively. The values of the matrix are $A_{ij} = r_i + c_j + n_{ij}$ where $n_{ij} \sim \mathcal{N}(0, 1)$. The third dataset GAUSSIAN contains 1000 points taken from a 10-dimensional normal distribution with unit variance and center drawn from $\mathcal{N}_{10}(0, 1)$. The fourth dataset COMPONENT contains 1000 random points with 5 intrinsic dimensions linearly transformed into a 50-dimensional space with Gaussian noise added to the points. The last artificial dataset CLUSTER has a clear Gaussian cluster structure with 10 clusters, each with 10–200 points. Cluster centers were drawn from $\mathcal{N}_{100}(0, 1)$ and cluster points were produced by adding random values from $\mathcal{N}_{100}(0, 1)$ to the cluster center. The datasets are available at http://www.cis.hut.fi/mrojala/randomization/.

We also used the gene expression data GENE by Scherf *et al.* [37]. The dataset contains gene expression measurements from 1375 genes in 60 human cancer cell lines. Around 2% of the values were missing and replaced by the average of the values in the corresponding rows. The second real dataset RETAIL is an aggregated version of retail market basket data [38]. The original dataset contained 88162 transactions with 16470 different products. Only products that occurred in at most 100 transactions were preserved, leaving 14632 products. Then we randomly partitioned the rows as well as the products into groups of 100. This gave an 881×146 matrix, where each entry was the total number of 1s in the 100×100 submatrix of the original data. The resulting matrix contained integer values from 0 to 11.

The values in all the datasets were linearly scaled to $[0, 1]$. Table 3 shows some properties of the datasets. In the following, rows correspond to data points and columns to dimensions.

Table 3. The number of rows and columns, and the average values and standard deviations of the values in the datasets.

Dataset	Rows	Columns	Mean	Std
RANDOM	100	100	0.473	0.132
SUM	100	100	0.500	0.146
GAUSSIAN	1000	10	0.529	0.142
COMPONENT	1000	50	0.278	0.116
CLUSTER	1117	100	0.509	0.081
GENE	1375	60	0.578	0.110
RETAIL	881	146	0.191	0.135

8.2. Selecting Parameter Values

We discuss various approaches for selecting the parameter values for our methods systematically. *GeneralMetropolis* depends on the constant $w > 0$, which controls the amount of difference allowed in the maintained statistics and affects the converge speed. *SwapDiscretized* has as parameters the number of row and column classes R and C. With all methods we also have to fix the number of attempts I. For other, less important parameters we use these fixed values in our experiments: In histogram-based difference measures we use 100 bins, effectively distinguishing values at 0.01 precision. In local modifications `Mask` and `Add`, we use the scale parameter $s = 0.1$, i.e., the addition is selected from $U([-0.1, 0.1])$.

We studied four different parameter selection procedures, where the parameter values are selected

1. by hand;
2. by restricting the differences in row and column statistics below some limit;
3. by requiring a minimum data-specific distance between randomized and original data;
4. by requiring convergence in a given number of attempts.

All these procedures have certain desirable and undesirable properties. Selecting the values by hand can produce good results. However, the problem is that tuning easily continues until desired results are obtained. In the second approach, deciding a good limit for the amount of difference allowed in the maintained statistics is problematic. However, we can, for example, choose the parameter w so that the average L_1 difference in cdf is around 0.01.

The third approach is based on the idea that the normalized root mean square distance between the original matrix and a random matrix is usually tightly distributed around some value in $[0, \sqrt{2}]$, when the matrices share their row and column statistics exactly. If we knew this distance, we could require the same distance from the randomized matrices and use this information for selecting appropriate w and I.

In this paper, we use the fourth approach. We fix the number of attempts I and find as tight a scaling coefficient w as possible such that the method still converges in I attempts. This allows a good comparison between the methods and produces reasonable results. Based on extensive empirical

tests with our methods we suggest that $I = 100mn$ attempts is a good general parameter value for *GeneralMetropolis*.

The appropriate parameter value w was determined by using an extended binary search approach: first find a range $[w_0, w_1]$ that contains the correct value. Then use binary search to find the correct value with 1% precision. As the convergence test, we monitored the normalized root mean square distance between the original and a randomized matrix. The method was assessed to be converged in I attempts if the distance between the original matrix and a randomized matrix produced with I attempts was at most 1% smaller than the distance between the original matrix and a randomized matrix produced with $2I$ attempts. The distances were calculated as median distances over 30 randomized matrices.

A similar approach could be used to select the number or row and column classes for *SwapDiscretized*. However, in this paper we will, in general, use $R = \lfloor 2\sqrt{n} \rfloor$ row classes and $C = \lfloor 2\sqrt{m} \rfloor$ column classes. For the dataset RETAIL we will use $R = C = 12$, since the dataset contains only 12 different values. The number of attempts I needed is found by monitoring the normalized root mean square distance and requiring that it converges. The procedure is repeated 30 times.

Tables 4 and 5 give the parameter values for *GeneralMetropolis* and *SwapDiscretized*, respectively. The parameter w has some variation between different combinations of difference measures and local modifications as each combination has different properties. We note that with the selected number of row and column classes, *SwapDiscretized* needs less attempts with all datasets than the $I = 100mn$ attempts *GeneralMetropolis* performs.

8.3. Convergence and Performance

We performed experiments to measure how well the methods maintain the statistics (distributions or means and variances), and what their performance is. We used Java implementations integrated with MATLAB on a 2.2 GHz Opteron with 4 GB of main memory. We will show some of the results only on the GENE dataset. However, similar convergence and performance results were obtained also with the artificial datasets.

Recall that the parameter w was selected for *GeneralMetropolis* by fixing the number of attempts to $I = 100mn$ and finding w such that the process still converges. Here we confirm that the methods are really able to converge in $I = 100mn$ steps with the selected scaling constants w. Figure 8 shows the normalized root mean square distance defined in Equation 10 as a function of attempts when randomizing the GENE data matrix. Each data point presented in the figure is a result from an independent single randomization. The randomization was started from the original matrix and was first run I steps backward and then I steps forward, where I is the number of attempts presented in the x-axis. Recall that the normalized root mean square distance does not measure the difference in the maintained row and column statistics, but the dissimilarity between the two matrices.

Table 4. Values of parameter w for *GeneralMetropolis* obtained for four combinations of difference measures and local modifications in different datasets. The methods use $I = 100mn$ attempts in all datasets.

	GM-CdfHist		*GM-MeanStd*	
Dataset	`Rotate`	`Resample`	`Mask`	`Rotate`
RANDOM	35.1	25.9	25.4	14.7
SUM	36.1	31.4	39.1	18.8
GAUSSIAN	40.4	23.7	34.6	17.3
COMPONENT	48.1	53.2	70.8	31.4
CLUSTER	46.1	28.9	25.4	12.2
GENE	37.1	26.1	24.8	12.3
RETAIL	23.6	17.6	17.6	12.3

Table 5. Values of parameters for *SwapDiscretized* for different datasets: the number of row classes R, the number of column classes C and the number of attempts I.

Dataset	R	C	I/mn
RANDOM	20	20	26
SUM	20	20	41
GAUSSIAN	6	63	11
COMPONENT	14	63	41
CLUSTER	20	66	32
GENE	15	74	51
RETAIL	12	12	17

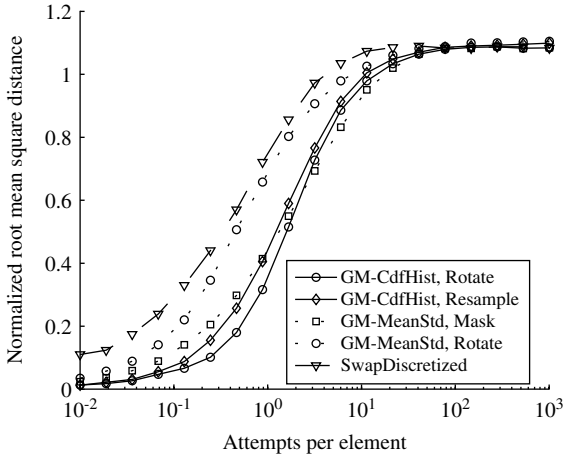


Fig. 8 The normalized root mean square distance between the original and randomized matrix as a function of attempts used to backward and forward run with GENE dataset. The x-axis gives the number of attempts per element, $I/(mn)$, on log-scale.

Table 6. Normalized root mean square distances between the original and randomized matrices. The values are average values over 999 samples. The standard deviation is in all cases less than 0.01.

	GM-CdfHist		*GM-MeanStd*			*Permutation*		
Dataset	`Rotate`	`Resample`	`Mask`	`Rotate`	*SwapDiscr.*	All	Row	Col
RANDOM	1.384	1.379	1.398	1.381	1.392	1.414	1.407	1.406
SUM	0.811	0.816	0.800	0.799	0.801	1.414	1.121	1.180
GAUSSIAN	0.810	0.829	0.845	0.835	0.833	1.414	1.382	0.930
COMPONENT	0.581	0.574	0.529	0.544	0.565	1.414	1.111	1.052
CLUSTER	1.359	1.360	1.388	1.376	1.372	1.414	1.408	1.389
GENE	1.078	1.085	1.081	1.094	1.083	1.414	1.153	1.385
RETAIL	1.361	1.363	1.375	1.356	1.377	1.414	1.397	1.395

From Figure 8 we observe that all methods converged to approximately the same distance. *SwapDiscretized* converged the fastest, but this is partly due to the different parameter selection procedure. The figure supports our empirical convergence experiments where $I = 100\ mn$ attempts are used for *GeneralMetropolis* and $I = 51\ mn$ attempts for *SwapDiscretized* with the GENE dataset.

Table 6 gives the average normalized root mean square distances between randomized and original matrices for different methods and datasets. The randomized samples are produced with the parameters given in Tables 4 and 5. The variation between different methods is small, whereas the variation in distances between different datasets is large. The distances in datasets RANDOM and PEAKS are close to $\sqrt{2} \approx 1.414$, the expected distance of a random permutation according to Theorem 8. On the other hand, in GAUSSIAN dataset the values in each column are tightly concentrated around one value. Thus, also the distances of randomized matrices that share the original row and column statistics differ a lot from $\sqrt{2}$. We notice that also the distances of the row-wise and column-wise permutations differ from $\sqrt{2}$, but the distances are always larger than the distances with our methods.

To confirm that the randomizations produced by a single method are significantly different, we calculated the pairwise normalized root mean square distances between 999 randomized samples for each method and dataset. The results are shown in Table 7. The pairwise distances in Table 7 almost equal the distances from the original matrices shown in Table 6. Thus, we may conclude that the methods indeed produce different randomizations in each run.

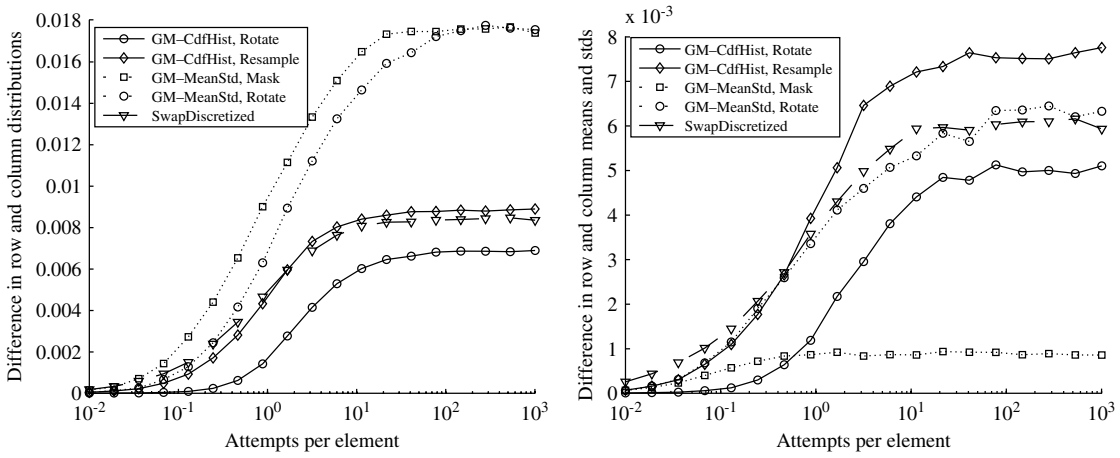
Table 8 shows the acceptance rate of attempts and the running times for the methods on GENE dataset. We notice that *GM-Cdf* is over ten times slower than *GM-CdfHist*. When used with larger matrices the relative time difference increases. The results of *GM-Cdf* and *GM-CdfHist* are in all cases very close to each other. Thus, there is no need to use the slower but slightly more accurate *GM-Cdf* with

Table 8. Performance of the methods with GENE dataset ($mn = 82500$). Acceptance rate is the number accepted local modifications divided by the number of attempted ones. Randomization was done with 100 *mn* attempts with all *GeneralMetropolis* methods and with 51 *mn* attempts with *SwapDiscretized*. Time is the time needed to produce one sample matrix.

Method	Acceptance rate	Time (s)
GM-Cdf, `Rotate`	0.237	223.53
GM-CdfHist, `Rotate`	0.182	16.86
GM-CdfHist, `Resample`	0.448	6.14
GM-MeanStd, `Mask`	0.309	9.67
GM-MeanStd, `Rotate`	0.241	10.24
SwapDiscretized	0.215	4.83
Permutation		0.06

Table 7. Pairwise normalized root mean square distances between randomized matrices. The presented values are average values over pairwise distances of 999 samples. The standard deviation is in all cases less than 0.01.

	GM-CdfHist		*GM-MeanStd*		
Dataset	`Rotate`	`Resample`	`Mask`	`Rotate`	*SwapDiscr.*
RANDOM	1.387	1.387	1.397	1.381	1.391
SUM	0.836	0.846	0.801	0.806	0.809
GAUSSIAN	0.823	0.857	0.843	0.841	0.834
COMPONENT	0.631	0.619	0.526	0.558	0.599
CLUSTER	1.360	1.364	1.366	1.359	1.371
GENE	1.085	1.100	1.074	1.102	1.086
RETAIL	1.363	1.366	1.376	1.356	1.377



(a) Difference in distributions

(b) Difference in means and standard deviations

Fig. 9 Convergence of the combined average row and column difference in the maintained statistics as a function of attempts used for randomizing GENE dataset.

GeneralMetropolis. The other methods are fast enough for practical use. Compared to producing a random permutation of the original matrix, the methods take approximately 100 times longer. This is due to the number of attempts $I = 100mn$ used.

Next we analyzed the difference in distributions between the original matrix and randomized matrices produced by the methods. In Figure 9 the convergence of the distribution difference 9(a) and the mean and standard deviation difference 9(b) are shown for the GENE dataset. The differences have converged in the same number of attempts as the distances in Figure 8. In Table 9 the average differences of rows and columns of randomized samples are shown for different methods on GENE dataset. From Figure 9 and Table 9 we can conclude that the methods that are designed to preserve distributions, *GeneralMetropolis* with *GM-CdfHist* and *SwapDiscretized*, can really preserve the distributions better than the methods that preserve only means and variances. On the other hand, the distribution-preserving methods can also preserve means and variances quite well.

For comparison, we have included also the differences in the maintained statistics with total, row-wise and column-wise permutations in Table 9. In all cases, the permutation methods produce much larger differences in the row and column statistics than our methods. Of course, row-wise permutations preserve the row statistics exactly, and similarly for column-wise permutations.

8.4. Significance Testing of Structural Measures

To test the methods in actual data analysis tasks, we used three structural measures: the maximum correlation value between matrix rows, the K-means clustering error with 10 clusters calculated with K-means++ algorithm [39], and the fraction of variance explained by the first five principal components. We generated 999 randomized

Table 9. Average difference in three statistics for rows and columns between the original matrix and the randomized matrices in GENE dataset. The distribution difference is the L_1 difference in cdfs. The mean and std differences are the absolute differences in the corresponding values. Values in the table are multiplied by 1000.

	Distribution		Mean		Std	
Method	Rows	Cols	Rows	Cols	Rows	Cols
GM-CdfHist, `Rotate`	7.02	1.19	2.11	0.20	3.04	0.36
GM-CdfHist, `Resample`	9.09	1.39	3.11	0.28	4.66	0.53
GM-MeanStd, `Mask`	17.77	9.50	0.00	0.00	0.91	0.05
GM-MeanStd, `Rotate`	17.74	5.95	2.23	0.06	4.21	0.06
SwapDiscretized	8.66	0.94	3.08	0.12	3.20	0.12
Permutation	58.82	21.69	46.62	17.75	38.93	14.77
Row-wise permutation	0.00	21.64	0.00	17.73	0.00	14.69
Column-wise permutation	58.76	0.00	46.54	0.00	38.92	0.00

matrices with each method for each dataset using the Besag approach introduced in Section 4.5. From these samples we calculated the average value and the standard deviation of the three structural measures. Finally, the empirical p-values were calculated for the original data.

The hypothesis for K-means was that the original data has smaller clustering error, and for correlation and principal components that the original data has a higher value of the corresponding structural measure than the randomized matrices. The results for the maximum correlation value are shown in Table 10, for the K-means clustering in Table 11, and for the principal components in Table 12. For comparison we have also included results with different permutation methods. Additionally, in Table 11 we give the standard deviations of K-means clustering errors.

With RANDOM dataset we observe that all methods have resulted in p-values near 0.5 for the three structural measures. Thus, as expected, the RANDOM dataset does not contain any significant structure.

The dataset SUM was generated with a simple procedure where each row and column has its own mean value in Gaussian distribution. Thus, it is expected that the dataset is explained by the row and column statistics. In Table 10 we see that all our methods have produced p-values that imply that the maximum correlation found in the dataset SUM is nonsignificant, whereas, all the permutation methods suggest that there is some structure in the data. It is also notable that the maximum correlation found in permuted matrices is much smaller than in the original matrix.

The dataset GAUSSIAN contains a high maximum correlation as shown in Table 10. To see that also high structural measures can be nonsignificant we used our methods to assess the significance of the maximum correlation in the GAUSSIAN dataset. With all our methods we get p-values near 0.4, implying that the maximum correlation is not surprising, given the row and column statistics. Actually, also the column-wise permutation considers the result as nonsignificant, which implies that the dataset is mainly described by the column distributions which is clear when considering how the dataset was generated.

The last two artificial datasets CLUSTER and COMPONENT were generated to contain significant structure. In Table 11 we have assessed the significance of the K-means clustering error in the dataset CLUSTER and in Table 12 the significance of the variance explained by the first five principal components in the dataset COMPONENT. We note that all methods have given a p-value 0.001 as the result. However, the methods that preserve both the row and column statistics have preserved the structural measure much more than the permutation methods. This effect is especially visible with the COMPONENT dataset (Table 12).

We used the methods also to study the significance of the three structural measures in a gene expression dataset GENE. We notice that the original structures disappear in the randomizations. However, our methods have again preserved the structure more than the permutation methods. We can conclude that the structures in the dataset GENE are independent from the row and column statistics, and therefore, interesting.

The second real dataset RETAIL was formed with an aggregation procedure. We assessed the significance of the principal component analysis on RETAIL dataset. The results in Table 12 show that according to our methods the intrinsic structure in the RETAIL dataset is a purely random artefact explained by the row and column distributions. Contrary to this, the permutation methods assessed the structural measure as significant.

8.5. Applying Randomization Methods in DNA Microarray Analysis

Next, we demonstrate the performance of our methods on a real data analysis task. We look at the complete dataset from [37], of which GENE was a filtered subset. The complete dataset contains gene expression measurements for

Table 10. Maximum correlation values between the rows calculated for original and randomized matrices with different methods. The average maximum correlations in 999 randomizations are given. The standard deviations of maximum correlations were less than 0.04 in all cases. The p-values are calculated for the original data matrices with the hypothesis that the original data contains a high correlation.

	RANDOM		SUM		GAUSSIAN		GENE	
Method	Measure	p-value	Measure	p-value	Measure	p-value	Measure	p-value
Original data	0.363		0.694		0.993		0.995	
GM-CdfHist, `Rotate`	0.359	0.391	0.674	0.133	0.992	0.399	0.688	0.001
GM-CdfHist, `Resample`	0.360	0.416	0.666	0.069	0.992	0.353	0.664	0.001
GM-MeanStd, `Mask`	0.360	0.405	0.699	0.587	0.992	0.411	0.656	0.001
GM-MeanStd, `Rotate`	0.360	0.397	0.692	0.415	0.992	0.372	0.649	0.001
SwapDiscretized	0.361	0.410	0.689	0.343	0.992	0.396	0.714	0.001
Permutation	0.360	0.389	0.360	0.001	0.972	0.007	0.615	0.001
Row-wise perm.	0.358	0.373	0.360	0.001	0.972	0.008	0.615	0.001
Column-wise perm.	0.360	0.414	0.582	0.001	0.992	0.395	0.658	0.001

Table 11. K-means clustering errors with 10 clusters calculated for original and randomized matrices with different methods. The average clustering errors in 999 randomizations are given. The values in parentheses are the standard deviations. The p-values are calculated for the original data matrices with the hypothesis that the original data contain cluster structure.

	RANDOM		CLUSTER		GENE	
Method	Measure	p-value	Measure	p-value	Measure	p-value
Original data	147.0		457.3		525.5	
GM-CdfHist, `Rotate`	146.7 (0.5)	0.677	659.2 (0.8)	0.001	605.2 (1.3)	0.001
GM-CdfHist, `Resample`	145.0 (0.7)	0.998	657.8 (1.0)	0.001	620.4 (1.7)	0.001
GM-MeanStd, `Mask`	147.0 (0.5)	0.534	654.1 (0.9)	0.001	591.1 (1.1)	0.001
GM-MeanStd, `Rotate`	146.7 (0.5)	0.702	656.5 (0.8)	0.001	621.1 (1.4)	0.001
SwapDiscretized	146.7 (0.5)	0.728	659.0 (0.8)	0.001	604.3 (1.3)	0.001
Permutation	147.0 (0.6)	0.484	688.2 (0.4)	0.001	924.3 (1.0)	0.001
Row-wise perm.	147.0 (0.6)	0.505	690.3 (0.6)	0.001	661.6 (0.9)	0.001
Column-wise perm.	146.7 (0.6)	0.683	660.8 (0.6)	0.001	859.0 (1.3)	0.001

Table 12. The fraction of variance explained by the first five principal components calculated for original and randomized matrices with different methods. The average fractions of variance in 999 randomizations are given. The standard deviations of fractions of variances were less than 0.003 in all cases. The p-values are calculated for the original data matrices with the hypothesis that the original data contains a high fraction of variance explained.

	RANDOM		COMPONENT		RETAIL		GENE	
Method	Measure	p-value	Measure	p-value	Measure	p-value	Measure	p-value
Original data	0.173		0.941		0.080		0.605	
GM-CdfHist, `Rotate`	0.174	0.596	0.687	0.001	0.079	0.261	0.440	0.001
GM-CdfHist, `Resample`	0.174	0.625	0.695	0.001	0.079	0.142	0.425	0.001
GM-MeanStd, `Mask`	0.174	0.651	0.776	0.001	0.080	0.409	0.456	0.001
GM-MeanStd, `Rotate`	0.174	0.606	0.747	0.001	0.079	0.303	0.422	0.001
SwapDiscretized	0.174	0.622	0.708	0.001	0.080	0.404	0.441	0.001
Permutation	0.174	0.653	0.140	0.001	0.064	0.001	0.115	0.001
Row-wise perm.	0.174	0.682	0.456	0.001	0.077	0.001	0.407	0.001
Column-wise perm.	0.174	0.604	0.203	0.001	0.066	0.001	0.149	0.001

9365 genes in 60 human cancer cell lines. As exemplified by the analysis in the original paper, gene expression data analysis always contains numerous steps. In the preprocessing phase, variables such as technical microarray quality measures, expression levels and their ratios are used to discard the majority of genes from further analysis.[1] The set of potentially interesting genes can be further reduced by applying statistical tests, or by clustering and focusing on clusters containing important genes. Typically, the final goal is to produce a set of 10 or so candidate genes, so that they can be individually tested in further wet laboratory experiments.

In their analysis, Scherf *et al.* filtered the set of genes according to different conditions for signal strength, and performed hierarchical clustering of the 60 cell lines. We showed in Section 8.4 that the GENE dataset contained significant cluster structure. However, that is not always the case in microarray data analysis. Due to rapid development in the field, very often many of the analysis methods are *ad hoc* in nature. Especially in early microarray studies, the set of genes was reduced with somewhat crude procedures.

We mimicked such *ad hoc* methods by creating an analysis pipeline that filters the set of genes and performs hierarchical clustering on the reduced set. For filtering, all expression matrix elements are discretized by giving them a value of 1 or 0, so that the elements with values in the highest quartile of expression values are set to 1 and the rest to 0.

The set of genes (i.e., rows) is next reduced by calculating the sum of 1's for each row. Rows are sorted by their sums, and 50 rows with highest sums are included in the filtered data. If there are multiple rows with same sums to be included, they are picked at random. Typically, the aim of the preprocessing phase is to arrive at a set that is small enough to be easily managed and manually analyzed further. Therefore, we chose to select only 50 genes. Finally, a row-wise hierarchical clustering using Euclidean distance is performed for the 50 rows, and the clustering error is used as the structural measure.

[1] Commonly approved standards for microarray data preprocessing do not exist, although there is a working group for data transformation and normalization under the MGED society http://genome-www5.stanford.edu/mged/normalization.html.

For comparison, we also repeat the filtering steps as described in Ref. [37], and report results for them. The original preprocessing was based partly on manual inspection of the raw scanned data, so our reproduction will follow only part of the original preprocessing steps. Data rows are filtered by identifying rows that have at least four values above or below manually tuned thresholds for over- and underexpression, and do not have more than four unknown values.

The *ad hoc* gene analysis pipeline was applied to the original data from [37]. The clustering errors and p-values for the original pipeline and the discretization pipeline are given in Table 13. We used randomization methods *GM-CdfHist* with `Rotate`, *GM-MeanStd* with `Mask` and *SwapDiscretized* as they were seen to produce good results for Euclidean distance-based clustering in Section 8.4, and for comparison we used the permutation method.

We require p-values to be below 0.05, because it is the most commonly used significance level in microarray analysis. We see that the simple permutation method fails to produce useful results. When permuting all values of the matrix, the cluster structure of the data matrix is destroyed, and therefore, results are estimated to be significant for our crude discretization preprocessing pipeline. On the other hand, the clustering error arising from the original pipeline has a p-value of 1.00. It is based on fixed bounds of signal strength. Permutation changes the data matrix so much that a very different number of rows are selected, thus comparing clustering errors does not produce meaningful results. More delicate randomization methods, however, preserve the structure of data more closely, but for significance testing it would be advisable to fix the number of rows to filter as was done in our discretization pipeline.

Significance results from our methods are more realistic. The original pipeline is labeled as significant, and the discretization-based pipeline is given p-values that indicate nonsignificance. Because the number of rows to be filtered was fixed at 50, it can be concluded that the discretization preprocessing failed to pick a subset of rows with significant cluster structure, i.e., when more realistic randomization was used to generate random gene expression datasets, cluster structures of similar strength were detected by chance.

Table 13. The p-values and clustering errors of Scherf *et al.* and discretization analysis pipelines with four different randomization methods. The results are calculated from 999 randomizations with the complete Scherf *et al.* data.

	Scherf *et al.*		Discretization	
Method	Meas.	p-val.	Meas.	p-val.
Original data	232.3		18.5	
Permutation	183.9	1.000	28.8	0.001
SwapDiscretized	303.7	0.001	20.7	0.114
GM-CdfHist, `Rotate`	324.5	0.001	21.7	0.060
GM-MeanStd, `Mask`	275.3	0.001	20.2	0.146

DNA microarrays were chosen as an example of real data, because discovering interesting structures in this high-throughput data is dependent on advanced data analysis techniques [40]. Multiphased analysis pipelines do not lend themselves to traditional methods of statistical significance testing, so there is a growing need for randomization-based significance testing in microarray data analysis. Microarrays have been criticized for their low signal-to-noise ratio, and there have been doubts that some microarray measurements do not necessarily contain any significant patterns at all. Our randomization experiments show that this is not the case with the dataset of Scherf *et al.*, but that with improper *ad hoc* methods the significant structure in the data can be destroyed. We also saw that the simple permutation methods do not produce very realistic random samples in DNA microarray significance testing. We hold the opinion that part of the blame that microarray measurements have been receiving should be addressed to inadequate robustness of data analysis methods.

9. PRACTICAL GUIDELINES FOR USING THE METHODS

In this section we discuss applying the methods in practice and give some useful guidelines. First, we compare the methods and their properties. After that we give some practical information on applying the methods on real problems.

9.1. Comparison of the Methods

We have introduced two main methods for randomizing a real-valued matrix, namely *GeneralMetropolis* and *SwapDiscretized*. The method *GeneralMetropolis* can use any difference measure and local modification, forming a few method combinations. We have studied difference measures *GM-Cdf*, *GM-CdfHist* and *GM-MeanStd* in more detail as well as local modifications `Resample`, `Rotate`, and `Mask`.

The only practical difference we noticed between difference measures *GM-Cdf* and *GM-CdfHist* was that *GM-CdfHist* is much faster for large datasets. The additional accuracy in *GM-Cdf* is negligible compared to the increase in randomization time. The time differences between *SwapDiscretized* and *GeneralMetropolis* with *GM-CdfHist* or *GM-MeanStd* were small. However, in general, *SwapDiscretized* seemed to be the fastest method. *GeneralMetropolis* with *GM-MeanStd* is somewhat faster

than *GeneralMetropolis* with *GM-CdfHist*. Also, the local modifications had an effect on the convergence time, the modifications based on small additions being slightly less efficient than others.

The main difference between the methods is the ability to preserve certain statistics. If only row and column means and variances need to be preserved, then *GeneralMetropolis* with *GM-MeanStd* and `Mask` is the best choice, since the means are preserved exactly and variances quite accurately as well. If the row and column distributions need to be preserved, then *SwapDiscretized* is an easy choice. Also *GeneralMetropolis* with *GM-CdfHist* and `Rotate` or `Resample` works well. The chosen local modification also affects the final value distributions.

The methods have different parameters. In general, selecting the scaling constant, w, with *GeneralMetropolis* is rather complicated. The effect of w is hard to understand directly. Contrary to this, *SwapDiscretized* has understandable parameters that can quite safely be selected by hand. The number of classes directly affects the difference in distributions. The other parameters regarding difference measures and local modifications do not have as much impact on the performance of the methods.

To conclude the comparison, all our methods have produced reasonable results in the experiments. Also, permutation methods were able to give good results in some cases depending on the nature of the dataset studied. However, in our opinion, *SwapDiscretized* is a good choice for general significance testing on real-valued matrices; also *GeneralMetropolis* with different difference measures and local modifications is sometimes a good choice.

9.2. Using the Methods in Practice

To make the practical use of our methods easier we have collected here some suggestions and guidelines. In most applications we suggest using *SwapDiscretized* due to its simplicity.

The first problem a user faces is selecting suitable parameter values. Background information about the data can be useful in selecting the parameters. For example, we may know that the values are only meaningful in precision 0.01 when the values are in $[0, 1]$. In such a case, it may be reasonable, for example, to use *SwapDiscretized* with $R = C = 100$ which corresponds to the precision 0.01. Another example is a contingency table where a natural discretization is directly available (as was the case with RETAIL dataset).

With *GeneralMetropolis* methods, exploiting the information of meaningful precision is a little bit harder than with *SwapDiscretized*. A corresponding difference scaling parameter w can be found with an extended binary search approach as discussed in Section 8.2. In general, we suggest fixing the precision beforehand and tuning the parameters to obtain the wanted precision. With *SwapDiscretized* the number of row and column classes can be calculated directly from this information.

After fixing the parameters controlling the differences in the maintained statistics, we should find an appropriate number of attempts I and assess the amount of randomness. The number of attempts I can be found, for example, by monitoring the convergence of the scaled root mean square distance or of a structural measure. It should, however, be noted that full convergence is not critical—the results will just be more conservative.

To check the randomness we can calculate the normalized root mean square distance between a randomized matrix and the original matrix. Qualifying whether a matrix is random enough is a hard question that is partly left for future work. Nevertheless, we can compare the distance to other distances obtained by permutation methods, but in general, those distances can be quite different. A heuristic approach is to start from a random permutation and run *GeneralMetropolis* with $w = \infty$ until convergence. The distance of the result from the original matrix may give some hint of the appropriate amount of randomness.

For the validity of p-values one has to use the Besag and Clifford approach introduced in Section 4.5. For comparison it is always a good idea to repeat the significance testing with permutation methods as well. The methods can sometimes be quite sensitive and deviations small, so one should also look at the difference in the structural measures and not only at the obtained p-value. The result can safely be assessed as significant with respect to row and column distributions if the difference in the structural measures is large and the p-value is near zero.

10. CONCLUDING REMARKS

We have considered the problem of randomization-based significance testing for data mining results on real-valued matrix data. We concentrated on methods for generating random matrices that have approximately the same row and column distributions, or means and variances, as the original data. Such randomized matrices can then be used to obtain empirical p-values for the data mining results.

We gave a general Metropolis algorithm that can be used with any difference measure and local modification as well as a simple method based on discrete swaps. The algorithms are based on different ways of iteratively updating a matrix. The algorithms work well, converging in a reasonable time. Our empirical results indicate that the obtained p-values clearly show whether the data mining result is significant or not.

There are obviously many open issues related to our current work. Starting from the algorithms, the methods

we presented are not the only possible ones. For example, the Metropolis scheme can be used with virtually any local modification for the matrices. It would be interesting to know the differences in the convergence of the methods in more detail.

As already the 0–1 case of our problem is computationally very hard, it seems quite difficult to obtain algorithms that would have a provable convergence time. Still, some more theoretical analyses would be welcome. Also, more study is needed in selecting the parameter values for our methods. Using parallel tempering with *GeneralMetropolis* could produce some speedup and accuracy, see, e.g., Ref. [41].

In applying the algorithms, we employed the schema suggested by Besag *et al.* [2,3,31]. This schema involves running the chain backward. The more efficient sequential versions of this approach would also be worth studying.

The final important open issue is what type of statistics should one try to preserve in randomization-based significance testing. The choice of the value distributions or the weaker first and second moments, means and variances, for the rows and columns seems fairly natural. Additionally, our approach is based on assuming that the rows and columns are independent of each other. However, also certain weaker or stronger statistics could be preserved. Preserving different statistics is an interesting topic for future work.

REFERENCES

[1] P. Good, Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses, New York, Springer, 2000.

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

[3] J. Besag. Markov chain Monte Carlo methods for statistical inference, http://www.ims.nus.edu.sg/Programs/mcmc/files/besag_tl.pdf, 2004.

[4] G. W. Cobb and Y. P. Chen, An application of Markov chain Monte Carlo to community ecology, Am Math Mon 110 (2003), 265–288.

[5] H. J. Ryser, Combinatorial properties of matrices of zeros and ones, Can J Math 9 (1957), 371–377.

[6] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas, Assessing data mining results via swap randomization, ACM Trans Knowl Discov Data 1(3) (2007).

[7] Y. Chen, P. Diaconis, S. P. Holmes, and J. S. Liu, Sequential Monte Carlo methods for statistical analysis of tables, J Am Stat Assoc 100(469) (2005), 109–120.

[8] I. Bezáková, N. Bhatnagar, and E. Vigoda, Sampling binary contingency tables with a greedy start, In Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, SIAM, Miami, Florida, 2006, 414–423.

[9] M. Dyer, Approximate counting by dynamic programming, In Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, San Diego, June 9-11 2003, 2003, 693–699.

[10] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs, Bioinformatics 20(11) (2004), 1746–1758.

[11] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (2002), 824–827.

[12] M. Ojala, N. Vuokko, A. Kallio, N. Haiminen, and H. Mannila, Randomization of real-valued matrices for assessing the significance of data mining results, In SDM '08: Proceedings of the SIAM International Conference on Data Mining, SIAM, Atlanta, Georgia, 2008, 494–505.

[13] S. Brin, R. Motwani, and C. Silverstein, Beyond market baskets: Generalizing association rules to correlations, In SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, ACM, Tucson, 1997, May 13- 15, 1997, 265–276.

[14] B. Liu, W. Hsu, and Y. Ma, Pruning and summarizing the discovered associations, In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, San Diego, 1999, August 15-18 1999, 125–134.

[15] B. Liu, W. Hsu, and Y. Ma, Identifying non-actionable association rules, In Knowledge Discovery and Data Mining, 2001, 329–334.

[16] H. Xiong, S. Shekhar, P. N. Tan, and V. Kumar, Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs, In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Seattle, 2004, August 22–25 2004, 334–343.

[17] N. Megiddo and R. Srikant, Discovering predictive association rules, In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York, AAAI Press, 1998, August 27-31, 1998, 274–278.

[18] W. DuMouchel and D. Pregibon. Empirical Bayes screening for multi-item associations, In Knowledge Discovery and Data Mining, 2001, 67–76.

[19] S. Jaroszewicz and D. A. Simovici, A general measure of rule interestingness, In 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001), Freiburg, Germany, 2001, 253–265.

[20] A. Ukkonen and H. Mannila, Finding outlying items in sets of partial rankings, In PKDD, 2007, 265–276.

[21] J. E. Moore and R. K. Swihart, Toward ecologically explicit null models of nestedness, Oecologia 152(4) (2007), 763–777.

[22] W. Ulrich and N. J. Gotelli, Disentangling community patterns of nestedness and species co-occurrence, Oikos 116(12) (2007), 2053–2061.

[23] W. Ulrich and N. J. Gotelli, Null model analysis of species nestedness patterns, Ecology 88(7) (2007), 1824–1831.

[24] M. J. Wonham and E. Pachepsky, A null model of temporal trends in biological invasion records, Ecology Letters 9(6) (2006), 663–672.

[25] A. Waldron, Null models of geographic range size evolution reaffirm its heritability, Am Nat 170(2) (2007), 221–231.

[26] P. Diaconis and A. Gangolli, Rectangular arrays with fixed margins, In Discrete Probability and Algorithms, 1995, 15–41.

[27] T. A. B. Snijders, Enumeration and simulation methods for 01511 matrices with given marginals, Psychometrika 56 (1991), 397–417.
[28] B. Y. Wang and F. Zhang, On the precise number of *(0,1)*-matrices in $U(R, S)$, Discrete Math 187 (1998), 211–220.
[29] I. Bezáková, A. Sinclair, D. Stefankovic, and E. Vigoda. Negative Examples for Sequential Importance Sampling of Binary Contingency Tables, http://arxiv.org/abs/math.ST/0606650, 2006.
[30] J. G. Sanderson, Testing ecological patterns, Am Sci 88 (2000), 332–339.
[31] J. Besag and P. Clifford, Sequential Monte Carlo p-values, Biometrika 78(2) (1991), 301–304.
[32] Y. Benjamini and Y. Hochberg, Controlling the false discovery rate: A practical and powerful approach to multiple testing, J R Stat Soc Series B Methodological 57(1) (1995), 289–300.
[33] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, M. Teller, and E. Teller, Equations of state calculations by fast computing machines, J Chem Phys 21 (1953), 1087–1092.
[34] W. K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, Biometrika 57 (1970), 97–109.
[35] S. Kullback and R. Leibler, On information and sufficiency, Ann Math Stat 22(1) (1951), 79–86.
[36] S. H. Cha and S. N. Srihari, On measuring the distance between histograms, Pattern Recognit 35(6) (2002), 1355–1370.
[37] U. Scherf, D. T. Ross, M. Waltham, L. H. Smith, J. K. Lee, L. Tanabe, K. W. Kohn, W. C. Reinhold, T. G. Myers, D. T. Andrews, D. A. Scudiero, M. B. Eisen, E. A. Sausville, Y. Pommier, D. Botstein, P. O. Brown and J. N. Weinstein, A gene expression database for the molecular pharmacology of cancer, Nat Genet 24 (2000), 236–244.
[38] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets, Using association rules for product assortment decisions: A case study, In Knowledge Discovery and Data Mining, 1999, 254–260.
[39] D. Arthur and S. Vassilvitskii, K-means++: the advantages of careful seeding, In SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 2007, 1027–1035.
[40] A. Brazma and J. Vilo, Gene expression data analysis, Microbes Infect 3 (2001), 823–829.
[41] D. J. Earl and M. W. Deem, Parallel tempering: theory, applications, and new perspectives, Phys Chem Chem Phys 7 (2005), 3910–3916.