

# Computing the Exact Distribution of the Extremes of Linear Combinations of Spacings <sup>\*</sup>

Fred W. Huffer

Department of Statistics  
Florida State University  
Tallahassee, FL 32306  
huffer@stat.fsu.edu

Chien-Tai Lin

Department of Mathematics  
Tamkang University  
Tamsui, Taiwan, R.O.C.  
chien@sparc20.math.tku.edu.tw

## Abstract

We develop a methodology for evaluating the distribution of the minimum or the maximum of linear combinations of spacings and then present some applications of this methodology. The main idea underlying our approach was given by Huffer (1988): A recursion is used to break up the joint distribution of several linear combinations of spacings into a sum of simpler components. We continue applying this recursion until we obtain components which are simple and easily expressed in closed form. In this paper we propose algorithms for the systematic application of this recursion. Our methods are fairly general and can be used to solve a variety of problems involving linear combinations of spacings or exponential random variables. A special case of great importance is when the linear combinations are simply sums of consecutive spacings. We develop specialized methods for dealing with this case. Because the output of our procedure is a polynomial whose coefficients are computed exactly, we can supply numerical answers which are accurate to any required degree of precision.

**Key words:** Cluster Probabilities; Scan Statistic; Symbolic Computation.

---

<sup>\*</sup>The authors would like to thank J. S. Wu for helpful suggestions and discussions relating to the material in Section 3.3.

# 1 INTRODUCTION

Let  $X_1, X_2, \dots, X_n$  be  $n$  points independently drawn from a uniform distribution on the interval  $[0, 1]$  and let  $X_{(1)} \leq X_{(2)} \leq \dots \leq X_{(n)}$  be the associated order statistics. The spacings  $S_1, S_2, \dots, S_{n+1}$  are defined to be the successive differences between the order statistics  $S_i = X_{(i)} - X_{(i-1)}$ , where we take  $X_{(0)} = 0$  and  $X_{(n+1)} = 1$ . Let  $\mathbf{S}$  denote the vector of spacings;  $\mathbf{S} = (S_1, S_2, \dots, S_{n+1})'$ . (Throughout this paper the letter  $n$  is reserved exclusively for denoting the number of random points  $X_i$ .)

For a set  $\Delta \subset \{1, 2, \dots, n+1\}$  define

$$S(\Delta) = \sum_{i \in \Delta} S_i.$$

Many important situations involve the joint distribution of overlapping sums of spacings, so that we are often required to calculate probabilities of the form

$$P\left(\bigcap_{i=1}^r \{S(\Delta_i) > d\}\right) \quad \text{or} \quad P\left(\bigcap_{i=1}^r \{S(\Delta_i) < d\}\right) \quad (1)$$

where the sets  $\Delta_1, \Delta_2, \dots, \Delta_r$  overlap. Probabilities of this type arise when computing the distribution of the scan statistic (see Naus (1965, 1966), Glaz (1989, 1992, 1993), Loader (1991)), when investigating the distribution of the number of clumps or clusters of the points  $X_i$  (see Glaz and Naus (1983), Huffer and Lin (1995)), and when studying the joint distribution of the  $m$ -spacings (see Cressie (1977), Holst (1980), Hall (1984, 1986), Glaz et al. (1994)).

In this paper we present a method for evaluating probabilities of the form (1). The computer programs we have written to implement this method are also convenient for evaluating quantities which can be expressed as sums of these probabilities. The approach we use is fairly general and can be used for a wide variety of configurations of the sets  $\Delta_i$ . Also, we emphasize that our programs can supply numerical answers which are accurate to any required degree of precision. That is because the final “answer” produced by our method is a (piecewise) polynomial, and not just a numerical value. The coefficients in this polynomial are computed exactly, so the polynomial can be stored and later evaluated by a symbolic computation package such as MAPLE.

The approach we use was sketched in Huffer (1988) and more fully developed in Lin (1993). It depends on repeated use of the recursion given in equations (2) and (5) below. This recursion is used to re-express a probability like that in (1) by decomposing

it into a sum of similar, but simpler components. The same recursion is then applied to each of these components and so on. The process is continued until we obtain components which are simple and easily expressed in closed form. In this paper we develop algorithms for the systematic application of this recursion.

This paper is organized as follows. In Section 2 we describe the recursion which is the basis of our approach. We also introduce some notation and properties we shall need in later sections. Section 3 describes our algorithms. We have developed two algorithms. One algorithm (presented in Section 3.3) computes the probabilities (1) in cases where each of the sets  $\Delta_i$  consists of a block of consecutive integers. This situation arises in problems involving the scan statistic or  $m$ -spacings. The other algorithm (presented in Section 3.2) can handle more general configurations of the sets  $\Delta_i$ , and can also solve some problems involving linear combinations of the spacings more complicated than simple sums like  $S(\Delta)$ . Section 3.1 contains remarks which pertain to both algorithms. This section describes the general approach used in our algorithms and the form of the answers which they produce. Section 4 gives three examples in which our methods are applied. This section can be read before reading the algorithm details in Sections 3.2 and 3.3.

## 2 Basic Properties and Definitions

Our approach is based on the following recursion.

Let  $\mathbf{\Gamma}$  be a  $r \times (n + 1)$  real matrix. Let  $\mathbf{S} = (S_1, S_2, \dots, S_{n+1})'$  be the random vector of spacings. Let  $\mathcal{P}(\mathbf{\Gamma})$  denote the probability measure of  $\mathbf{\Gamma}\mathbf{S}$  so that  $P(\mathbf{\Gamma}\mathbf{S} \in B) = (\mathcal{P}(\mathbf{\Gamma}))(B)$ . For any  $\boldsymbol{\xi} \in \mathcal{R}^k$ , define  $\mathbf{\Gamma}_{i,\boldsymbol{\xi}}$  to be the  $r \times (n + 1)$  matrix obtained by replacing the  $i^{th}$  column of  $\mathbf{\Gamma}$  by  $\boldsymbol{\xi}$ . The basic recursion is the following.

**Theorem 1** *Suppose  $\mathbf{c} = (c_1, c_2, \dots, c_{n+1})'$  satisfies  $\sum_{i=1}^{n+1} c_i = 1$ . Let  $\boldsymbol{\xi} = \mathbf{\Gamma}\mathbf{c}$ . Then*

$$\mathcal{P}(\mathbf{\Gamma}) = \sum_{i=1}^{n+1} c_i \mathcal{P}(\mathbf{\Gamma}_{i,\boldsymbol{\xi}}). \quad (2)$$

This recursion was first obtained by Michelli (1980) as a result about multivariate B-splines. It was rediscovered in the context of spacings by Huffer (1988). The recursion also holds when  $\mathbf{S}$  is a vector of i.i.d. exponential random variables. This was shown by Lin (1993). Examples showing how the basic recursion is applied in computations can be found in Huffer (1988) and Lin (1993).

## Notation

In this paper we are not interested in the entire joint distribution  $\mathcal{P}(\mathbf{\Gamma})$ , but only in quantities of the form (1). We now introduce the notation we shall use for these problems.

Let  $\mathbf{A}$  be any matrix having at most  $n + 1$  columns. Take  $\mathbf{\Gamma}$  to be the matrix with  $n + 1$  columns obtained by padding  $\mathbf{A}$  with columns of zeros;  $\mathbf{\Gamma} = (\mathbf{A} | \mathbf{0})$ . Let  $r$  be the number of rows in  $\mathbf{A}$  and define  $\mathbf{Y} = (Y_1, Y_2, \dots, Y_r)$  by  $\mathbf{Y} = \mathbf{\Gamma}\mathbf{S}$ . For any value of  $d$ , we define

$$\{\mathbf{A}\}_d^1 = P\{Y_i > d \text{ for all } i\} = P\{\min Y_i > d\}, \quad (3)$$

and

$$\{\mathbf{A}\}_d^2 = P\{Y_i < d \text{ for all } i\} = P\{\max Y_i < d\}. \quad (4)$$

When the value of  $d$  is held fixed in an argument, we delete the subscript and just write  $\{\mathbf{A}\}^1$  or  $\{\mathbf{A}\}^2$ . We also omit the superscript when convenient.

The quantity  $\{\mathbf{A}\}$  is well defined so long as the number of spacings  $n + 1$  is greater than or equal to the number of columns in  $\mathbf{A}$ . The value of  $\{\mathbf{A}\}$  depends, of course, on  $n$ , but we do not indicate this in the notation because the value of  $n$  is not important in most of our manipulations.

The vector  $(S(\Delta_1), S(\Delta_2), \dots, S(\Delta_r))'$  can be written as  $\mathbf{\Gamma}\mathbf{S}$  where  $\mathbf{\Gamma} = (\Gamma_{ij})$  is the  $r \times (n + 1)$  binary matrix with entries  $\Gamma_{ij} = 1$  if  $j \in \Delta_i$  and  $\Gamma_{ij} = 0$  otherwise. Thus, in evaluating probabilities of the form (1), we need consider only binary matrices  $\mathbf{A}$  in (3) and (4). However, our methods can also be applied in many cases where  $\mathbf{A}$  is *not* a binary matrix. These non-binary cases arise naturally in some problems. The discussion which follows applies to general matrices  $\mathbf{A}$ ; we shall not restrict ourselves to the binary case until Section 3.3.

When specialized to problems of the type (3) and (4), the basic recursion (2) becomes the following:

*Let  $\mathbf{A}$  be a matrix having  $p$  columns with  $p \leq n + 1$ . Suppose  $\mathbf{c} = (c_1, c_2, \dots, c_p)'$  satisfies  $\sum_{i=1}^p c_i = 1$ . Let  $\boldsymbol{\xi} = \mathbf{A}\mathbf{c}$ . Then*

$$\{\mathbf{A}\} = \sum_{i=1}^p c_i \{\mathbf{A}_{i,\boldsymbol{\xi}}\}. \quad (5)$$

It is understood that all of the braces  $\{\cdot\}$  appearing in (5) have a common superscript of 1 or 2 and a common subscript of  $d$ . When using this recursion we shall often refer to the matrix  $\mathbf{A}$  as the “mother”, and the matrices  $\mathbf{A}_{i,\xi}$  (perhaps after being simplified by the properties given below in (6)) as the “daughters”. We also use the same mother/daughter terminology when referring to the values  $\{\mathbf{A}\}$  and  $\{\mathbf{A}_{i,\xi}\}$ .

### Useful Properties

We now list some properties which are useful in the process of evaluating  $\{\mathbf{A}\}$ . These properties are straightforward and their proofs are omitted.

Properties S1–S4 given below allow us to rearrange and simplify the matrix  $\mathbf{A}$  in various ways. We state these properties in terms of evaluating  $\{\mathbf{A}\}^1$ .

(S1) if the  $i^{\text{th}}$  row of  $\mathbf{A}$  dominates (is componentwise greater than or equal to) the  $j^{\text{th}}$  row, the  $j^{\text{th}}$  row can be deleted without changing the value of  $\{\mathbf{A}\}^1$ .

The value of  $\{\mathbf{A}\}^1$  remains the same when

(S2) a column of zeros is deleted,

(S3) the columns are permuted, (6)

(S4) the rows are permuted.

The same properties apply to  $\{\mathbf{A}\}^2$  except that (S1) must be reversed. In other words,

(S1') if the  $i^{\text{th}}$  row of  $\mathbf{A}$  is dominated by (is componentwise less than or equal to) the  $j^{\text{th}}$  row, the  $i^{\text{th}}$  row can be deleted without changing the value of  $\{\mathbf{A}\}^2$ .

Properties S2 and S3 are consequences of the fact that the spacings  $S_1, S_2, \dots, S_{n+1}$  are exchangeable random variables. Properties S1 and S1' just amount to saying that redundant inequalities can be deleted.

If  $\mathbf{A}$  is a block diagonal matrix, then we can evaluate  $\{\mathbf{A}\}$  by decomposing each of the blocks independently. We shall use this property in Section 3. To state this property more precisely, we introduce the notation  $\mathbf{B} \odot \mathbf{C}$  to represent a block diagonal matrix with blocks  $\mathbf{B}$  and  $\mathbf{C}$ , that is,

$$\mathbf{B} \odot \mathbf{C} = \begin{pmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{pmatrix}. \quad (7)$$

where  $\mathbf{0}$  denotes a matrix of zeros. Now we can write:

$$\text{If } \{\mathbf{B}\} = \sum_i \lambda_i \{\mathbf{B}_i\}, \text{ then } \{\mathbf{B} \odot \mathbf{C}\} = \sum_i \lambda_i \{\mathbf{B}_i \odot \mathbf{C}\}. \quad (8)$$

This is valid so long as the decomposition  $\{\mathbf{B}\} = \sum_i \lambda_i \{\mathbf{B}_i\}$  is obtained through application of the recursion (5). Also note that, for matrices having block diagonal form, properties S3 and S4 above allow us to permute the ordering of the blocks so that, for example,  $\{\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}\} = \{\mathbf{B} \odot \mathbf{A} \odot \mathbf{C}\} = \{\mathbf{A} \odot \mathbf{C} \odot \mathbf{B}\}$ .

### Explicit Formulas

For explicit calculations, we rely on the following formula which is given in Theorem 2.1 of Khatri and Mitra (1969). Let  $\Delta_1, \Delta_2, \dots, \Delta_r$  be nonempty subsets of  $\{1, 2, \dots, n+1\}$  with cardinalities  $|\Delta_i| = 1 + \ell_i$ . Define  $\boldsymbol{\ell} = (\ell_1, \dots, \ell_r)$ . Suppose  $0 < d_i < 1$  for  $i = 1, \dots, r$ . If the sets  $\Delta_i, i = 1, \dots, r$  are disjoint, then

$$P\left(\bigcap_{i=1}^r \{S(\Delta_i) > d_i\}\right) = \sum_{\mathbf{0} \leq \mathbf{k} \leq \boldsymbol{\ell}} \binom{n}{k_1, \dots, k_r} \left(1 - \sum_{i=1}^r d_i\right)_+^{n - \sum_{i=1}^r k_i} \prod_{i=1}^r d_i^{k_i}. \quad (9)$$

where  $\mathbf{k} = (k_1, \dots, k_r)$  is an  $r$ -tuple of integers and  $\mathbf{0} \leq \mathbf{k} \leq \boldsymbol{\ell}$  means that  $0 \leq k_i \leq \ell_i$  for all  $i$ . Here we use  $(x)_+$  to denote the positive part, that is,  $(x)_+ = \max(x, 0)$ . For our purposes, a very convenient way to rewrite this formula is as follows. For integers  $j \geq 0$  and real values  $\lambda \geq 0$ , define

$$R(j, \lambda) = \begin{cases} \binom{n}{j} d^j (1 - \lambda d)^{n-j} & \text{for } \lambda d < 1, \\ 0 & \text{for } \lambda d \geq 1. \end{cases} \quad (10)$$

The dependence of  $R$  on  $n$  and  $d$  can be left implicit because these values are fixed in any given application of our methods. In terms of  $R$ , formula (9) becomes: If the sets  $\Delta_i, i = 1, \dots, r$  are disjoint and the values  $\beta_i, i = 1, \dots, r$  are positive, then

$$P\left(\bigcap_{i=1}^r \{S(\Delta_i) > \beta_i d\}\right) = \sum_{\mathbf{0} \leq \mathbf{k} \leq \boldsymbol{\ell}} \binom{\sum_i k_i}{k_1, \dots, k_r} \left(\prod_{i=1}^r \beta_i^{k_i}\right) R(\sum_i k_i, \sum_i \beta_i). \quad (11)$$

To evaluate  $\{\mathbf{A}\}$ , we continue decomposing matrices using (5) until we reach “simple” terms which can be evaluated using (11). Writing our answers in terms of  $R$  allows us to obtain very compact expressions for  $\{\mathbf{A}\}$  by suppressing the dependence on  $n$  and  $d$ . (See the examples in Section 4.)

## 3 Algorithms

### 3.1 Remarks on Overall Structure

Given a large matrix  $\mathbf{A}$ , our goal is to devise algorithms for evaluating  $\{\mathbf{A}\}$  by repeated application of the recursion (5). Applying the recursion once to  $\{\mathbf{A}\}$  produces a set of “daughters”  $\{\mathbf{B}\}$ . Upon repeated applications of the recursion, each of these daughters in turn acquires its own set of daughters, etc. Thus the natural way to view the process of evaluating  $\{\mathbf{A}\}$  is as the construction of a tree whose bottom-most daughters (or terms) are all “simple”. For our purposes, a term  $\{\mathbf{B}\}$  is considered simple if it can be directly evaluated using equation (11). Collecting together the bottom-most simple terms gives a representation of the form

$$\{\mathbf{A}\} = \sum_i w_i \{\mathbf{B}_i\}. \quad (12)$$

Now, evaluation of the simple terms  $\{\mathbf{B}_i\}$  using (11) leads to a formula for  $\{\mathbf{A}\}$  as a sum of piecewise polynomials:

$$\{\mathbf{A}\} = \sum_i c_i R(j_i, \lambda_i). \quad (13)$$

The final answers produced by our methods are expressions of the form (13).

When we are discussing the evaluation of  $\{\mathbf{A}\}$ , we shall think in terms of building a tree as described above. In implementing our algorithms, we do not actually maintain a tree structure of mothers, daughters, granddaughters, etc. This would be inefficient. In constructing such a tree, the same terms  $\{\mathbf{B}\}$  would occur repeatedly. We would not want to store and separately evaluate a large number of identical terms; it makes more sense to combine identical terms. For this reason, we have chosen to store all of our intermediate results (our work in progress) in three lists: a list  $\mathcal{N}$  of terms which are *not* simple and require further decomposition, a list  $\mathcal{S}$  of simple terms which need no further work, and a list  $\mathcal{F}$  of terms for which we are unable to find a suitable decomposition, that is, terms for which our methods fail. A step in our procedure consists of removing the first term in  $\mathcal{N}$  and using the recursion (5) to decompose it into simpler daughter terms. The daughter terms are each examined. Any that are “simple” or “failures” are added to the lists  $\mathcal{S}$  or  $\mathcal{F}$  respectively. The remaining non-simple terms are added to the list  $\mathcal{N}$ . Whenever a new term is added to  $\mathcal{S}$ ,  $\mathcal{N}$  or  $\mathcal{F}$ ,

we first check to see if this term is identical to any term already on the list. If so, the new term is combined with the old term. If not, the new term is placed into the list in a suitable position. These lists are organized to facilitate searching. In particular, the list  $\mathcal{N}$  is ordered (at least roughly) by the complexity of the terms, with the complexity of a term  $\{\mathbf{B}\}$  defined primarily in terms of the dimensions of  $\mathbf{B}$ . This ensures that the “harder” terms are evaluated first.

Our decomposition procedure terminates when the list  $\mathcal{N}$  is exhausted. Ideally, upon termination the list  $\mathcal{F}$  is empty. In this case, our result is the list of simple terms  $\mathcal{S}$  which represents a decomposition of the form in (12).

We shall now describe two algorithms. The first algorithm attempts to evaluate  $\{\mathbf{A}\}$  for any matrix whose entries are all rational numbers. We use the word “attempt” because this algorithm will sometimes fail; there will be terms in the list  $\mathcal{F}$  upon termination of the algorithm. The second algorithm is more specialized; it is designed to evaluate  $\{\mathbf{A}\}$  only for matrices  $\mathbf{A}$  belonging to a certain class of binary matrices which arises frequently in applications. This second algorithm is guaranteed to succeed; the list  $\mathcal{F}$  is not needed in the implementation of this algorithm.

### 3.2 Algorithm for General Matrices with Rational Entries

Let  $\mathbf{A}$  be an arbitrary matrix with rational entries. We shall now describe a general approach for evaluating  $\{\mathbf{A}\}$ . Our description will be somewhat sketchy for two reasons. First, a complete description of the algorithm would be very lengthy. Secondly, the algorithm remains under development. The current version of the algorithm can evaluate  $\{\mathbf{A}\}$  for many cases of interest, but fails in others. We are working on improvements to the algorithm. Part of the reason for giving at least a sketchy description of this general approach is to contrast it with the more satisfactory state of affairs achieved by the specialized algorithm described in the next section. A much more detailed explanation of the general algorithm may be found in Lin (1993).

Our approach for evaluating  $\{\mathbf{A}\}$  is roughly as follows. We look through a short list of “plausible” vectors  $\boldsymbol{\xi}$  (which we call the *targets*). For each target  $\boldsymbol{\xi}$ , we try to find vectors of coefficients  $\mathbf{c}$  which *hit* the target, that is, which solve the equations  $\boldsymbol{\xi} = \mathbf{A}\mathbf{c}$  with  $\sum_i c_i = 1$  as required by the recursion (5). For each such solution, we examine the daughter matrices produced by the recursion. If all of these daughters are “simpler”

than their mother  $\mathbf{A}$ , we declare that we are finished with the decomposition (or reduction) of  $\{\mathbf{A}\}$ ; we now add these daughters to our tree and proceed to decompose them by the same process. If any of the daughters fails to be “simpler” than their mother, we reject this solution and look for further solution vectors  $\mathbf{c}$ . When we can find no more solution vectors  $\mathbf{c}$ , we move on to the next target vector  $\boldsymbol{\xi}$  and try again. (In our computer programs, if we exhaust our list of targets without finding a suitable decomposition of  $\{\mathbf{A}\}$ , we give up and add  $\{\mathbf{A}\}$  to the list  $\mathcal{F}$ . Then we go on to evaluate the next term in  $\mathcal{N}$  (if any terms remain).) In summary, our approach is to generate a variety of possible vectors  $\boldsymbol{\xi}$  and  $\mathbf{c}$ , examine the daughters produced by using these vectors in the recursion (5), and accept (that is, use) the first choice we find for which all the daughters are “simpler” than the mother.

To understand this procedure, we must explain what is meant by the terms “simpler” and “plausible” in the previous paragraph. We discuss this in the next paragraphs.

We call a matrix  $\mathbf{A}$  “simple” if we can evaluate  $\{\mathbf{A}\}$  directly using (11). A daughter matrix  $\mathbf{B}$  is considered “simpler” than its mother  $\mathbf{A}$  if we think it will require fewer decompositions (applications of the recursion) to reduce it to simple matrices. This vague notion must be given some precise definition in our computer programs. What is the best definition of “simpler”? To some extent, this question must be addressed empirically, and the answer may vary somewhat depending on the situation. The main ingredients in our current definition are the following:

We consider  $\mathbf{B}$  to be simpler than  $\mathbf{A}$

- (a) if the dimension of  $\mathbf{B}$  (the number of rows and columns) is smaller than that of  $\mathbf{A}$ , or
- (b) if  $\mathbf{A}$  and  $\mathbf{B}$  have the same dimension, but the number of nonzero entries in  $\mathbf{B}$  is less than in  $\mathbf{A}$ .

If there are “ties” in these two conditions we do the following.

- (c) We count the number of nonzero entries in each row of  $\mathbf{B}$  and  $\mathbf{A}$ . If the vector of counts for  $\mathbf{B}$  strictly majorizes that for  $\mathbf{A}$  in the sense of Schur, we say that  $\mathbf{B}$  is simpler than  $\mathbf{A}$ .

Conditions (a) and (b) are easy to understand. The concept of “majorization” used in (c) is discussed in Hardy, Littlewood, and Pólya (1952) and Marshall and Olkin (1979). Condition (c) was introduced because using it tends to create rows which have either a small or large number of nonzero entries. Then, in future generations, we are more likely to get daughter matrices which have a dominated row which reduces the dimension of the matrix by the property S1 (or S1’) in (6).

The definition of “simpler” given above is somewhat ad hoc and can probably be improved upon. A good definition of “simpler” will help us to evaluate  $\{\mathbf{A}\}$  in an efficient manner, using a relatively small number of decompositions (5). But in any case, some definition of “simpler” (even if it is *not* a very good one) is needed to prevent the occurrence of infinite loops. An infinite loop occurs when one of the descendants of  $\mathbf{A}$  (a daughter or grand-daughter or great-grand-daughter or ...) is identical to  $\mathbf{A}$ . Any reasonable definition of “simpler” will prevent this by keeping the decomposition process moving in a consistent direction.

A target vector  $\boldsymbol{\xi}$  is “plausible” if we think that using it is likely to cause simplification, that is, produce daughters which are simpler than their mother. We shall now describe the list of plausible targets used in our current program. The composition of this list is somewhat arbitrary and may be subject to future revision. First on our list is the zero vector  $\mathbf{0}$ . If we can hit this target, then property S2 in (6) guarantees that all of the daughters will have fewer columns than their mother. (Frequently, the number of rows can also be reduced by subsequent application of property S1 or S1’.) Let  $\mathbf{e}_i$  be the vector which is 1 in the  $i$ -th entry and zero in all others. If we fail to hit the zero vector, the next targets that we try are vectors of the form  $\lambda \mathbf{e}_i$  with  $\lambda \neq 0$ . Hitting the target  $\lambda \mathbf{e}_i$  is likely to create daughters with fewer nonzero entries than their mother. The columns  $\mathbf{a}_i$  of  $\mathbf{A}$  are the next set of plausible targets that we try. If we can hit an appropriately chosen column  $\mathbf{a}_i$ , then the daughters of  $\mathbf{A}$  will have fewer distinct columns than their mother. This makes it more likely that the daughters will have dominated rows (which reduces the dimension by property S1 in (6)) or will satisfy the majorization condition (c) above. Other plausible targets are vectors of the form  $\lambda \mathbf{e}_i + \mu \mathbf{e}_j$  which are zero in all coordinates except the  $i^{th}$  and  $j^{th}$ , etc. Hitting these targets will again tend to create daughters with fewer nonzero entries than their mother.

An important part of the general method outlined above is the procedure we use to find, for a given target  $\boldsymbol{\xi}$ , a vector  $\mathbf{c}$  which satisfies both  $\mathbf{A}\mathbf{c} = \boldsymbol{\xi}$  and  $\sum_i c_i = 1$ . These two conditions may be written more succinctly as  $\mathbf{A}^*\mathbf{c} = \boldsymbol{\xi}^*$  where  $\mathbf{A}^*$  and  $\boldsymbol{\xi}^*$  are obtained from  $\mathbf{A}$  and  $\boldsymbol{\xi}$  by appending a row of 1's and a single 1 respectively. The procedure we use to solve the equations  $\mathbf{A}^*\mathbf{c} = \boldsymbol{\xi}^*$  is a modification of the well known method based on the LU decomposition of the matrix  $\mathbf{A}^*$ . A description of this "LU method" may be found in Press et al. (1986). The available algorithms for the LU method are usually carried out using real numbers and assume that the inverse of the matrix  $\mathbf{A}^*$  exists so that the solution of the equation  $\mathbf{A}^*\mathbf{c} = \boldsymbol{\xi}^*$  is unique. We have modified the LU method in two respects. First, we implement the procedure using rational arithmetic with rational numbers represented by pairs of integers. Thus, the solutions  $\mathbf{c}$  that we obtain are exact. Secondly, we must use the procedure in situations where a solution may not exist or may not be unique (i.e. the matrix  $\mathbf{A}^*$  may be singular or may not even be a square matrix). We have modified the procedure so that it will always yield a solution when one or more solutions exist, and will indicate if we are in a situation where no solutions exist.

### 3.3 Algorithm for Binary Matrices with Consecutive Ones

In this section we describe a specialized algorithm for computing  $\{\mathbf{A}\}$  when  $\mathbf{A}$  is a binary matrix where the 1's in each row form a contiguous block. This is an important class of matrices which arises naturally in many problems involving the scan statistic,  $m$ -spacings, and the probabilities of various types of clusters and gaps. Using our specialized algorithm, we can solve these problems much more quickly than is possible using the more general approach described earlier.

#### Notation

Before describing our algorithm we must introduce some notation. Let  $\mathcal{B}$  be the class of all binary matrices in which the 1's in each row form a contiguous block. An  $r \times p$  matrix  $\mathbf{A} = (A_{ij})$  in  $\mathcal{B}$  may be described by the pairs  $(a_i, b_i)$ ,  $i = 1, \dots, r$ , which specify the position of the first and last 1 in each row. For  $i = 1, \dots, r$ , define  $\Delta_i = \{j : a_i \leq j \leq b_i\}$ . Then we can write  $A_{ij} = 1$  if and only if  $j \in \Delta_i$ .

The matrices we can handle with our algorithm are essentially those in  $\mathcal{B}$ , but we can

actually restrict our attention to a smaller class of matrices  $\mathcal{C}$  which we now define. An  $r \times p$  matrix  $\mathbf{A}$  belongs to  $\mathcal{C}$  if and only if  $\mathbf{A} \in \mathcal{B}$  and the values  $(a_i, b_i)$ ,  $i = 1, \dots, r$ , which describe  $\mathbf{A}$  satisfy  $1 = a_1 < a_2 < \dots < a_r$ ,  $b_1 < b_2 < \dots < b_r = p$ , and  $a_{i+1} \leq b_i + 1$  for  $i = 1, \dots, r - 1$ . The reason we can restrict our attention to  $\mathcal{C}$  is that any matrix in  $\mathcal{B}$  can be reduced to a matrix in  $\mathcal{C}$  by employing the simplification properties S1 (or S1'), S2 and S4 given in (6), that is, for any  $\mathbf{A} \in \mathcal{B}$  there exists  $\mathbf{A}^* \in \mathcal{C}$  satisfying  $\{\mathbf{A}\} = \{\mathbf{A}^*\}$ . The dimensions of  $\mathbf{A}^*$  will be less than or equal to those of  $\mathbf{A}$ . The process of reducing  $\mathbf{A}$  to  $\mathbf{A}^*$  is straightforward. By permuting the rows, we may ensure that  $a_1 \leq a_2 \leq \dots \leq a_r$ . Then, in any cases with  $a_i = a_{i+1}$ , we must have either  $\Delta_i \subset \Delta_{i+1}$  or  $\Delta_{i+1} \subset \Delta_i$ , so that property S1 or S1' can be used to eliminate either row  $i$  or row  $i + 1$ . In a similar fashion we can ensure that  $b_1 < b_2 < \dots < b_r$ . Now, by deleting any columns of zeros, we can guarantee that  $a_1 = 1$ ,  $b_r = p$ , and  $a_{i+1} \leq b_i + 1$  for  $i \leq r - 1$ . We may think of the matrices in  $\mathcal{C}$  as being arranged in a canonical “descending” form.

It is convenient to introduce two more classes  $\mathcal{D}$  and  $\mathcal{E}$  of binary matrices both of which are contained in  $\mathcal{C}$ . Let  $\mathbf{A}$  be an  $r \times p$  matrix in  $\mathcal{C}$ . The class  $\mathcal{D} \subset \mathcal{C}$  consists of those matrices in which consecutive rows always overlap, that is,  $\mathbf{A}$  belongs to  $\mathcal{D}$  if and only if  $\Delta_i \cap \Delta_{i+1} \neq \emptyset$  for all  $i \leq r - 1$ . Let  $\mathcal{E} \subset \mathcal{C}$  consist of those matrices in which all of the rows are disjoint, that is,  $\mathbf{A}$  belongs to  $\mathcal{E}$  if and only if  $\Delta_i \cap \Delta_j = \emptyset$  whenever  $i \neq j$ . Matrices with only one row represent a degenerate case for the above definitions. The classes  $\mathcal{C}$ ,  $\mathcal{D}$ ,  $\mathcal{E}$  all coincide for matrices of this type; a  $1 \times p$  matrix belongs to  $\mathcal{C}$  (and also to  $\mathcal{D}$  and  $\mathcal{E}$ ) if and only if all the entries are 1.

Note that every matrix in  $\mathcal{C}$  either belongs to  $\mathcal{D}$  or can be expressed in a block diagonal form where each of the blocks belongs to  $\mathcal{D}$ . That is,

$$\begin{aligned} &\text{if } \mathbf{A} \in \mathcal{C}, \text{ there exist matrices } \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_b \\ &\text{in } \mathcal{D} \text{ such that } \mathbf{A} = \mathbf{A}_1 \odot \mathbf{A}_2 \odot \dots \odot \mathbf{A}_b. \end{aligned} \tag{14}$$

(If  $\mathbf{A} \in \mathcal{D}$ , then  $\mathbf{A} = \mathbf{A}_1$  and  $b = 1$ .) Here we have used the notation  $\odot$  in (7).

The matrices in  $\mathcal{E}$  are all matrices we consider to be simple: If  $\mathbf{A} \in \mathcal{E}$ , then  $\{\mathbf{A}\}$  is given in a simple closed form by equation (11) with  $\beta_i = 1$  for all  $i$ .

We can now describe our algorithm. The basic properties of this algorithm will be summarized in the following theorem.

**Theorem 2** *For any matrix  $\mathbf{A} \in \mathcal{C}$ , the algorithm given below in (15) and (18) uses the recursion (5) finitely many times to express  $\{\mathbf{A}\}$  in the form (12) where the values*

$w_i$  are integers and the matrices  $\mathbf{B}_i$  belong to  $\mathcal{E}$ . Moreover, all the daughter matrices encountered during the decomposition process belong to  $\mathcal{C}$ . Expanding each of the terms  $\{\mathbf{B}_i\}$  using (11), we obtain our final answer in the form (13) where the values  $c_i$  and  $\lambda_i$  are all integers.

For matrices in  $\mathcal{C}$ , the specialized algorithm described below is greatly superior to the more general algorithm in the previous section. This specialized algorithm is much faster than the general algorithm. This is because using (15) eliminates the trial and error search for an appropriate vector  $\mathbf{c}$  for use in (5) which is required in the general algorithm. The specialized algorithm also uses less memory than the general algorithm. That is because all the daughter matrices generated by the specialized algorithm belong to  $\mathcal{C}$ ; to store an  $r \times p$  matrix in  $\mathcal{C}$  we need only store the pairs  $(a_i, b_i)$  for  $i = 1, \dots, r$ . Finally, we note that when applying (5) in the general algorithm, the entries in  $\mathbf{A}$ ,  $\mathbf{c}$ , and  $\boldsymbol{\xi}$  are all rational numbers represented by pairs of integers, whereas in the specialized algorithm these entries are all integers. Thus, the specialized algorithm involves only addition and multiplication of integers, whereas the specialized algorithm requires us to implement a slightly cumbersome rational arithmetic.

### Decomposition of Matrices in $\mathcal{D}$

Given a matrix  $\mathbf{A} \in \mathcal{D}$ , we shall now describe how to apply the recursion (5) once to produce daughter matrices which are all “simpler” than  $\mathbf{A}$ . That is, we shall give an explicit description of the vectors  $\mathbf{c}$  and  $\boldsymbol{\xi}$  needed in (5), and then examine the resulting daughter matrices  $\mathbf{A}_{i,\xi}$ . This procedure for decomposing matrices in  $\mathcal{D}$  forms the core of our algorithm.

Let  $\mathbf{A} \in \mathcal{D}$  be an  $r \times p$  matrix with  $r \geq 2$ . We shall construct a vector  $\mathbf{c} = (c_1, c_2, \dots, c_p)'$  according to the procedure specified in (15) below. This procedure clearly produces a vector  $\mathbf{c}$  satisfying  $\sum_i c_i = 1$ . Let  $m^*$  denote the terminating value of  $m$ . There are two cases.

$$\begin{aligned} \text{(Case I)} \quad & \text{If } m^* < r, \text{ then } \mathbf{A}\mathbf{c} = \mathbf{0}. \\ \text{(Case II)} \quad & \text{If } m^* = r, \text{ then } \mathbf{A}\mathbf{c} = \mathbf{e}_r \\ & \text{and } \mathbf{A}_{\cdot i} \neq \mathbf{e}_r \text{ for all } i \text{ with } c_i \neq 0. \end{aligned} \tag{16}$$

Here  $\mathbf{e}_r$  is the vector with a 1 in the  $r$ -th position and 0 for all other entries, and  $\mathbf{A}_{\cdot i}$  is the  $i$ -th column of  $\mathbf{A}$ . The properties in (16) will be verified shortly.

<pre> Initialize:     c<sub>1</sub> := 1, c<sub>i</sub> := 0 for i ≥ 2     m := 1 Repeat:     IF m = r THEN STOP     ELSE {         c<sub>b<sub>m</sub></sub> := -1         c<sub>b<sub>m</sub>+1</sub> := +1         IF ∃ j &gt; m such that a<sub>j</sub> = b<sub>m</sub> + 1             THEN m := j             ELSE STOP     } </pre>	(15)
--	------

Here is an informal description of the procedure in (15). We call two rows  $i$  and  $j$  “adjacent” if  $b_i + 1 = a_j$ . We “mark” a row (say row  $i$ ) by setting  $c_k$  to be  $+1$  and  $-1$  in the positions  $k$  corresponding to the first and last nonzero entries in row  $i$ , that is, setting  $c_{a_i} = +1$  and  $c_{b_i} = -1$ . Our procedure consists of marking adjacent rows. We start with the first row and continue marking adjacent rows as long as possible. When we cannot continue, we terminate appropriately by adding a  $+1$  entry immediately after the  $-1$  entry indicating the end of the last marked row. An example will make this clear. What follows is a typical matrix  $\mathbf{A}$  in  $\mathcal{D}$  and the corresponding vector  $\mathbf{c}$ . We have underlined the adjacent rows that get marked by our procedure.

$$\mathbf{A} = \begin{pmatrix}
 \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1} & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{1} & \underline{1} & \underline{1}
 \end{pmatrix} \quad (17)$$

$$\mathbf{c} = \left( 1 \ 0 \ 0 \ 0 \ -1 \ 1 \ 0 \ 0 \ 0 \ -1 \ 1 \ 0 \ 0 \ 0 \ 0 \ -1 \ 1 \ 0 \ 0 \ 0 \right)$$

This example illustrates Case I of (16).

It is easy to convince oneself of the properties listed in (16) by working a few examples. We give a brief argument for these properties. Let  $\mathcal{J} = \{j : c_j \neq 0\}$  and recall that  $\Delta_i = \{j : A_{ij} = 1\}$ . Let  $v_i = |\mathcal{J} \cap \Delta_i|$ . Define  $\boldsymbol{\xi} = (\xi_i)$  to be the vector  $\boldsymbol{\xi} = \mathbf{A}\mathbf{c}$  and consider the value of  $\xi_i = \sum_{j \in \Delta_i} c_j$ . Since the nonzero entries

in  $\mathbf{c}$  alternate between 1 and  $-1$ , and the set  $\Delta_i$  consists of a block of consecutive integers, the only possible values of  $\xi_i$  are 0, 1 and  $-1$ . Moreover,  $\xi_i \neq 0$  if and only if  $v_i$  is an odd number. Suppose that  $\xi_i \neq 0$ . We cannot have  $v_i \geq 3$  because that would imply that  $\Delta_i$  *strictly* contained  $\Delta_k$  for some “marked” row  $k$ . (For matrices in  $\mathcal{D}$  there cannot be rows  $k, \ell$  ( $k \neq \ell$ ) with  $\Delta_k \subset \Delta_\ell$ .) Therefore,  $v_i$  must equal 1. Note that every  $-1$  entry in  $\mathbf{c}$  is positioned at the last nonzero entry of one of the “marked” rows. Also, every  $-1$  entry in  $\mathbf{c}$  is followed immediately by a  $+1$  entry, that is,  $c_j = -1$  implies  $c_{j+1} = +1$  for all  $j$ . This rules out the possibility that  $\xi_i = -1$  because the combination of  $v_i = 1$  and  $\xi_i = -1$  would then imply that  $\Delta_i$  is *strictly* contained within  $\Delta_k$  for some “marked” row  $k$ . We are left only with the case where  $\xi_i = 1$  and  $v_i = 1$ . The only way this can arise is if  $\mathcal{J} \cap \Delta_i = \max\{j : j \in \mathcal{J}\}$  (any other possibility leads to  $\Delta_i$  being *strictly* contained within  $\Delta_k$  for some “marked” row  $k$ ), but this means that row  $i$  is “adjacent” to the last “marked” row. This contradicts the termination conditions of the procedure in (15) unless row  $i$  is in fact the last row (the  $r$ -th row) of the matrix  $\mathbf{A}$  and  $m^* = r$ . In conclusion,  $\xi_i = 0$  except when we are in Case II of (16) and  $i = r$ , and then  $\xi_i = 1$ . This concludes the argument.

For  $\mathbf{A}$  in  $\mathcal{D}$ , let us examine the daughter matrices obtained by decomposing  $\{\mathbf{A}\}$  according to the procedure in (15). Our object is to note some properties of these matrices we shall need later in proving that our algorithm terminates and verifying the other properties listed in Theorem 2. Let  $\mathcal{J} = \{j : c_j \neq 0\}$ . The daughter matrices are  $\mathbf{A}_{i,\xi}$  for  $i \in \mathcal{J}$  where  $\xi = \mathbf{0}$  in Case I and  $\xi = \mathbf{e}_r$  in Case II. In Case I, we use property S2 in (6) to delete the  $i$ -th column of  $\mathbf{A}_{i,\xi}$ . The resulting matrix has one less column than  $\mathbf{A}$  and will clearly have contiguous 1’s in each row, that is, it will belong to  $\mathcal{B}$ . However, it may no longer be in  $\mathcal{D}$  as there may now be rows  $j$  for which  $a_j = a_{j+1}$  or  $b_j = b_{j+1}$  or  $\Delta_j \cap \Delta_{j+1} = \emptyset$ . But recall that every matrix in  $\mathcal{B}$  is equivalent to a reduced matrix in  $\mathcal{C}$  which has the same or smaller dimensions. Replacing daughters by these reduced matrices when necessary, we obtain a set of daughter matrices  $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_k$  with  $k = |\mathcal{J}|$  which all (i) belong to  $\mathcal{C}$ , (ii) have fewer columns than  $\mathbf{A}$ , and (iii) have the same or smaller number of rows as  $\mathbf{A}$ .

Now consider Case II. We can use property S3 of (6) to move the  $i$ -th column of  $\mathbf{A}_{i,\xi}$  into the last position. There are now two possibilities: the resulting matrix (call it  $\mathbf{B}$ ) either belongs to  $\mathcal{D}$  or it does not. We shall refer to these as cases II(a) and

II(b) respectively. We now deal with Case II(a). Let us compare the number of 1's in each row of  $\mathbf{A}$  and  $\mathbf{B}$ . Define  $u_j$  and  $u'_j$  to be the number of 1's in row  $j$  of  $\mathbf{A}$  and  $\mathbf{B}$  respectively. From (16), it is easy to see that  $u_r \leq u'_r$  and  $u_j \geq u'_j$  for  $j = 1, 2, \dots, r-1$  with strict inequality for at least one  $j$ .

Finally, consider Case II(b). The matrix  $\mathbf{B}$  clearly belongs to the class  $\mathcal{B}$  and is therefore equivalent to some matrix in  $\mathcal{C}$ ; call this matrix  $\mathbf{C}$ . Since  $\mathbf{B}$  does not belong to  $\mathcal{D}$ , there must be at least one row  $j$  in  $\mathbf{B}$  for which  $a_j = a_{j+1}$  or  $b_j = b_{j+1}$  or  $\Delta_j \cap \Delta_{j+1} = \emptyset$ . If either  $a_j = a_{j+1}$  or  $b_j = b_{j+1}$  holds, then the reduction to the equivalent matrix  $\mathbf{C}$  will use property S1 (or S1') at least once, and consequently  $\mathbf{C}$  will have fewer rows than  $\mathbf{B}$ . If  $\Delta_j \cap \Delta_{j+1} = \emptyset$ , then  $\mathbf{B}$  is a block diagonal matrix with at least two blocks. This implies that  $\mathbf{C}$  also has at least two blocks. Combining this with fact (14), we see that  $\mathbf{C}$  is a block diagonal matrix whose blocks belong to  $\mathcal{D}$  and have fewer rows than the parent matrix  $\mathbf{A}$ . In summary, after appropriate simplification, a daughter matrix in Case II(b) will belong to  $\mathcal{C}$ , and will either have fewer rows than  $\mathbf{A}$  or consist of blocks (belonging to  $\mathcal{D}$ ) having fewer rows than  $\mathbf{A}$ .

### Decomposition of Matrices in $\mathcal{C}$

We can now give a complete statement of our algorithm. Recall that, if a matrix can be written in a block diagonal form, then property (8) allows us to decompose that matrix by working separately on each block. Let  $\mathbf{A}$  be any matrix in  $\mathcal{C}$ . Write  $\mathbf{A}$  in the block diagonal form  $\mathbf{A} = \mathbf{A}_1 \odot \mathbf{A}_2 \odot \dots \odot \mathbf{A}_b$  where the blocks  $\mathbf{A}_i$  belong to  $\mathcal{D}$ . We compute  $\{\mathbf{A}\}$  as follows:

1. If  $\mathbf{A} \in \mathcal{E}$ , do nothing. (We are done. Evaluate  $\{\mathbf{A}\}$  using (11). )
2. Otherwise, locate the first block  $\mathbf{A}_i$  having at least two rows and decompose  $\mathbf{A}_i$  using the procedure for matrices in  $\mathcal{D}$ . (18)
3. Now apply this same process to all the daughter matrices obtained in the previous step.

We shall now verify the properties of this algorithm which were stated in Theorem 2. As described earlier, executing this algorithm leads (at least in principle) to the construction of a tree. All of the daughter matrices encountered during the construction of this tree belong to  $\mathcal{C}$ . This follows from our earlier examination of the daughter

matrices which result from decomposing a matrix in  $\mathcal{D}$  using (15); all of these daughter matrices, after appropriate simplification, belong to  $\mathcal{C}$ . Assume, for the moment, that the algorithm in (18) will always terminate in finitely many steps. (This is proved below.) It is clear that the only possible terminal nodes of the tree are matrices in  $\mathcal{E}$ ; any matrix in  $\mathcal{C}$  which does not belong to  $\mathcal{E}$  can be further decomposed by the algorithm. Also, because the procedure in (15) produces vectors  $\mathbf{c}$  whose nonzero entries  $c_i$  are either  $+1$  or  $-1$ , all of the terminal nodes in our tree will be multiplied by coefficients of  $+1$  or  $-1$ . Collecting together the terminal nodes then leads to a sum in which the coefficients  $w_i$  in (12) are all integers. For a matrix  $\mathbf{B}$  in  $\mathcal{E}$ , evaluating  $\{\mathbf{B}\}$  using (11) requires  $\beta_i = 1$  for all  $i$ . Thus, when evaluating (12) to obtain (13), all the coefficients  $c_i$  and  $\lambda_i$  will be integers.

To show that  $\{\mathbf{A}\}$  can be evaluated in finitely many applications of the recursion, we must show that the tree has finite depth. This means we must show that the tree does *not* contain any infinite sequences  $\mathbf{B}_1 \rightarrow \mathbf{B}_2 \rightarrow \mathbf{B}_3 \rightarrow \dots$  where  $\mathbf{A} \equiv \mathbf{B}_1$  and  $\mathbf{B}_i \rightarrow \mathbf{B}_{i+1}$  means that  $\mathbf{B}_{i+1}$  is one of the daughters produced by applying the algorithm to  $\mathbf{B}_i$ . We shall argue by contradiction. Suppose there exists such an infinite sequence. The algorithm proceeds by decomposing the diagonal blocks of a matrix from left to right. In carrying out the algorithm, that part of a matrix which has already been simplified remains fixed from then on, that is, if we can write  $\mathbf{B}_k$  in the form  $\mathbf{B}_k = \mathbf{E} \odot \mathbf{C}_k$  where  $\mathbf{E} \in \mathcal{E}$ , then for  $i \geq k$  we have  $\mathbf{B}_i = \mathbf{E} \odot \mathbf{C}_i$  for some matrix  $\mathbf{C}_i$  with dimensions no bigger than  $\mathbf{C}_k$ . From this it is clear that an infinite sequence can arise only if at some point the algorithm encounters (during step 2 of (18)) a block  $\mathbf{D} \in \mathcal{D}$  which cannot be reduced. More precisely, an infinite sequence can arise only if there exists  $\mathbf{D} \in \mathcal{D}$  with  $r \geq 2$  rows such that evaluating  $\{\mathbf{D}\}$  leads to the construction of a tree which contains an infinite sequence  $\mathbf{D} \equiv \mathbf{D}_1 \rightarrow \mathbf{D}_2 \rightarrow \mathbf{D}_3 \rightarrow \dots$  of matrices  $\mathbf{D}_i$  which all have  $r$  rows and belong to  $\mathcal{D}$ . (Again  $\mathbf{D}_i \rightarrow \mathbf{D}_{i+1}$  means that  $\mathbf{D}_{i+1}$  is one of the daughters produced by applying the algorithm to  $\mathbf{D}_i$ .) By the discussion in the previous section, each of the arrows “ $\rightarrow$ ” must represent one of the Cases I or II(a). (Case II(b) led to daughter matrices which either had fewer rows than their mother or did not belong to  $\mathcal{D}$ .) We shall refer to  $\mathbf{D}_i \rightarrow \mathbf{D}_{i+1}$  as the  $i$ -th transition in our sequence. Each transition belonging to Case I reduces the number of columns by one, so there can be only finitely many transitions of this type in our sequence. Thus, from

some point  $m$  on, all transitions must belong to Case II(a). Let  $u_{i,j}$  be the number of 1's in the  $j$ -th row of  $\mathbf{D}_i$ . For  $i \geq m$ , we know that  $u_{i,j} \geq u_{i+1,j}$  for  $j = 1, 2, \dots, r-1$  with *strict* inequality for at least one  $j$ . Thus, the number of transitions beyond  $m$  cannot exceed the total number of 1's in the first  $r-1$  rows of  $\mathbf{D}_m$ . This contradicts the existence of the infinite sequence and completes the proof.

## 4 Examples

This section contains three examples of computations carried out using the algorithms described in Section 3. The first two examples use the specialized algorithm (for binary matrices in  $\mathcal{C}$ ) described in 3.3. The last example illustrates use of the more general algorithm in 3.2. All three examples involve computing quantities which arise naturally when studying the clustering of random points on an interval or a circle. The computer programs used in these examples are available from the authors. The problems in the examples have been chosen to be small enough so that the answers fit conveniently on the page; our programs can handle substantially larger problems.

### Example 1

For our first example we evaluate  $\{\mathbf{A}\}_d^1$  for the  $10 \times 15$  matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \quad (19)$$

This matrix belongs to the class  $\mathcal{C}$  so that an expression for  $\{\mathbf{A}\}_d^1$  can be rapidly obtained by the algorithm in Section 3.3. This algorithm leads to an expression of the form (12) which is re-expressed as a piecewise polynomial of the form (13) which represents the final answer. The intermediate expression (12) is very lengthy in this case (and in many others), so we shall not give it. Our final answer is

$$\{\mathbf{A}\}_d^1 = 24596R(0,2) + 2002R(1,2) - 1716R(3,2) + 924R(4,2)$$

$$\begin{aligned}
& - 24506R(0,3) - 26658R(1,3) - 28550R(2,3) - 28882R(3,3) \quad (20) \\
& - 26574R(4,3) - 21822R(5,3) - 15436R(6,3) - 8826R(7,3) \\
& - 3636R(8,3) - 816R(9,3) - 89R(0,4) - 26R(1,4) - 4R(2,4)
\end{aligned}$$

We may now use this expression to compute  $\{\mathbf{A}\}_d^1$  for arbitrary  $n$  and  $d$ . (Of course, the quantity  $\{\mathbf{A}\}_d^1$  does not make sense unless the number of spacings is at least 15 which implies  $n \geq 14$ .) When  $n = 16$ , the quantity  $\{\mathbf{A}\}_d^1$  is simply the probability that *no* interval of length  $d$  contains more than 6 of the random points  $X_1, X_2, \dots, X_{16}$ . That is,  $1 - \{\mathbf{A}\}_d^1$  is the probability that there exists a cluster of 7 or more points in an interval of length  $d$ . Such clustering probabilities have been extensively studied particularly in connection with the distribution of the scan statistic, a test for the presence of non-random clustering. The literature is extensive; we will indicate just a few of the relevant papers. Much work has been done on the exact calculation of these clustering probabilities; see Neff and Naus (1980). Other work contains approximations (see Glaz (1989, 1992)) and asymptotic results (see Loader (1991)). Neff and Naus (1980) give tables of these clustering probabilities. As a check on our work in this paper, we note that (20) leads to answers in agreement with these tables. The notation used by Neff and Naus differs from ours; computing  $1 - \{\mathbf{A}\}_d^1$  using (20) with  $n = 16$  and  $d = 0.400, 0.399, \dots, 0.004, 0.003$  gives the values of  $P(n; N, p)$  reported in Table 1a of Neff and Naus (1980) for the case  $n = 7, N = 16$ .

The expression (20) is fairly typical of the answers produced by our methods. When the matrix  $\mathbf{A}$  is large, the expression (13) may contain hundreds of terms with large numbers of both positive and negative terms. Moreover, the weights  $c_i$  appearing in (13) vary dramatically in their magnitudes, including both very large positive and very large negative values. Thus, if the computations are done in standard single or double precision, there is the possibility that drastic cancellations between positive and negative terms will destroy the accuracy of our results. For this reason, we use the mathematical package MAPLE for the final computations. MAPLE allows us to specify an arbitrarily high degree of precision in our calculations. Thus, our procedure is implemented in two parts: a C program constructs the list of piecewise polynomial terms appearing in (13). Then these polynomial terms are passed to a MAPLE program for evaluation.

In Section 2 it was mentioned that the fundamental recursion (2) also holds when

$S_1, S_2, \dots, S_{n+1}$  are i.i.d. exponential random variables. In addition, the formula (11) continues to hold in this case so long as we redefine the function  $R$  to be

$$R(j, \lambda) = \frac{d^j}{j!} e^{-\lambda d}.$$

(This formula is appropriate for exponential random variables with mean 1.) Thus, all of the answers produced by our methods (that is, expressions of the form (13)) can also be interpreted as results about linear combinations of exponential random variables. In particular, the expression (20) gives the distribution of the minimum of a certain moving average process constructed from exponential random variables.

### Example 2

In this example we suppose there are  $n = 10$  random points on the unit interval. Fix a value of  $d$ . We shall say that a “clump” exists when there exists an interval of length  $d$  containing at least 4 of the 10 points; this occurs when  $S_i + S_{i+1} + S_{i+2} < d$  for some  $i$  in the range  $2 \leq i \leq 8$ . Define the number of clumps  $Y$  by

$$Y = \sum_{i=2}^8 Z_i \quad \text{where} \quad Z_i = I\{S_i + S_{i+1} + S_{i+2} < d\}.$$

We can use our methods to compute the moments of  $Y$ . To illustrate this we shall obtain an expression for  $E(Y^3)$ .

In this example, we are restricting ourselves to  $n = 10$  and to clumps of size 4. This is merely for convenience of presentation. Huffer and Lin (1995) discuss computing moments of the number of clumps in more general situations. They are interested in using these moments to construct approximations to the distribution of the scan statistic. The number of clumps  $Y$  has some importance in its own right. Glaz (1993) suggests that  $Y$  would be a reasonable statistic for testing uniformity; if the value of  $Y$  is much larger than expected, we reject the hypothesis of uniformity. We can use the moments of  $Y$  to compute bounds and approximations for the distribution of  $Y$  and hence obtain approximate critical values for this test. Glaz and Naus (1983), Dembo and Karlin (1992), Roos (1993), and Glaz et al. (1994) have studied the distribution of  $Y$ . Much recent work has been devoted to investigating the accuracy of Poisson and compound Poisson approximations to the distribution of  $Y$ .

Just for this example, we use  $\mathbf{M}_{j_1, j_2, \dots, j_r}$  to denote a binary matrix with  $r$  rows in which the  $k^{\text{th}}$  row consists of  $j_k$  leading zeros followed by a block of 3 consecutive ones.

The remaining entries in each row are zero. To make this notation clear we give two examples:

$$\mathbf{M}_{2,4} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{M}_{0,1,3} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

In terms of this notation, we have

$$\begin{aligned} E(Y^3) &= \sum_{i=2}^8 \sum_{j=2}^8 \sum_{k=2}^8 E(Z_i Z_j Z_k) \\ &= \sum_i E(Z_i) + 6 \sum_{i<j} E(Z_i Z_j) + 6 \sum_{i<j<k} E(Z_i Z_j Z_k) \\ &= \sum_i \{\mathbf{M}_i\}_d^2 + 6 \sum_{i<j} \{\mathbf{M}_{i,j}\}_d^2 + 6 \sum_{i<j<k} \{\mathbf{M}_{i,j,k}\}_d^2 \\ &= 7\{\mathbf{M}_0\} + 36\{\mathbf{M}_{0,1}\} + 30\{\mathbf{M}_{0,2}\} + 60\{\mathbf{M}_{0,3}\} \\ &\quad + 30\{\mathbf{M}_{0,1,2}\} + 24\{\mathbf{M}_{0,1,3}\} + 24\{\mathbf{M}_{0,2,3}\} + 18\{\mathbf{M}_{0,2,4}\} \\ &\quad + 72\{\mathbf{M}_{0,1,4}\} + 36\{\mathbf{M}_{0,2,5}\} + 6\{\mathbf{M}_{0,3,6}\} \end{aligned} \quad (21)$$

The final expression in (21) was obtained after rearranging terms using properties S2–S4 in (6) and then combining equal terms. The quantities  $\{\mathbf{M}_{i,j}\}_d^2$  and  $\{\mathbf{M}_{i,j,k}\}_d^2$  occurring in this expression can be evaluated by the algorithm in Section 3.3 because the matrices involved all belong to the class  $\mathcal{C}$ . Carrying out this evaluation leads to the final answer

$$\begin{aligned} E(Y^3) &= 343 + 41R(0,1) - 457R(1,1) - 889R(2,1) - 414R(0,2) - 402R(1,2) \\ &\quad + 306R(2,2) + 1434R(3,2) + 1998R(4,2) + 30R(0,3) + 162R(1,3) \\ &\quad + 378R(2,3) + 504R(3,3) + 108R(4,3) - 540R(5,3) - 540R(6,3). \end{aligned} \quad (22)$$

We note that our programs have been designed to conveniently evaluate sums like those in (21), that is, they can accept as “input” a list of terms representing a sum of the form  $\sum_i w_i \{\mathbf{A}_i\}$ .

### Example 3

For our last example we evaluate  $\{\mathbf{A}\}_d^2$  where

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (23)$$

This matrix has a pattern similar to that in (19) except that the entries in the last few rows “wrap around”. Because of this, the matrix does *not* belong to the class  $\mathcal{C}$  and we must use the general algorithm described in Section 3.2 to evaluate  $\{\mathbf{A}\}_d^2$ . This algorithm leads to the final answer

$$\begin{aligned} \{\mathbf{A}\}_d^2 &= 1 - 31455R(0, 2) + 196830R(0, 7/3) - 165376R(0, 5/2) \\ &\quad + 13750R(1, 2) - 21870R(1, 7/3) - 8960R(1, 5/2) \\ &\quad + 10R(2, 1) - 4080R(2, 2) - 10R(3, 1) + 600R(3, 2) \\ &\quad + 150R(4, 2) - 240R(5, 2) + 80R(6, 2). \end{aligned} \quad (24)$$

The expression (24) can be evaluated for  $n \geq 9$ . When  $n = 9$ , the value of  $\{\mathbf{A}\}_d^2$  can be interpreted as a “clustering” or “multiple coverage” probability for random points on a circle. When  $n = 9$ , the spacings  $S_1, S_2, \dots, S_{10}$  can be viewed as the spacings between 10 random points on a circle with circumference equal to 1. For 10 random points on a circle,  $\{\mathbf{A}\}_d^2$  is the probability that every arc of length  $d$  contains at least 4 of these points. (Note that, with probability 1, *every* arc of length  $d$  contains at least 4 points if and only if every closed arc of length  $d$  beginning at one of the random points  $X_i$  contains at least 5 points.) An equivalent interpretation is the following. Suppose 10 arcs of length  $d$  are placed at random on a circle. Then  $\{\mathbf{A}\}_d^2$  is the probability that every point on the circle is covered by at least 4 of these arcs. These multiple coverage probabilities have been studied by Holst (1980).

## References

- Cressie, N., The minimum of higher order gaps, *Austral. J. Statist.* **19** (1977) 132–143.
- Dembo, A., and S. Karlin, Poisson approximation for r-scan processes, *Ann. Appl. Probab.* **2** (1992) 329–357.
- Glaz, J., and J. Naus, Multiple clusters on the line, *Commun. Statist. – Theor. Meth.* **12** (1983) 1961–1986.
- Glaz, J., Approximations and bounds for the distribution of the scan statistic, *J. Amer. Statist. Assoc.* **84** (1989) 560–566.
- Glaz, J., Approximations for tail probabilities and moments of the scan statistic, *Comput. Statist. Data Anal.* **14** (1992) 213–227.
- Glaz, J., Approximations for the tail probabilities and moments of the scan statistic, *Statist. in Med.* **12** (1993) 1845–1852.
- Glaz, J., J. Naus, M. Roos, and S. Wallenstein, Poisson approximations for the distribution and moments of ordered m-spacings. *J. Appl. Probab.* **31A** (1994) 271–281.
- Hall, P., Limit theorems for sums of general functions of m-spacings, *Math'l. Proc. of the Cambridge Philosophical Soc.* **96** (1984) 517–532.
- Hall, P., On powerful distributional tests based on sample spacings, *J. Multiv. Anal.* **19** (1986) 201–224.
- Hardy, G. H., J.E. Littlewood, and G. Pólya, *Inequalities*, 2nd ed. (Cambridge University Press, Cambridge, 1952).
- Holst, L., On multiple covering of a circle with random arcs, *J. Appl. Probab.* **17** (1980) 284–290.
- Huffer, F., Divided differences and the joint distribution of linear combinations of spacings, *J. Appl. Probab.* **25** (1988) 346–354.
- Huffer, F., and C.T. Lin, Approximating the distribution of the scan statistic using moments of the number of clumps, (submitted for publication, 1995).
- Khatri, C.G., and S.K. Mitra, Some Identities and Approximations Concerning Positive

- and Negative Multinomial Distributions, in: P. R. Krishnaiah (Ed.), *Multivariate Analysis – II* (Academic Press, New York, 1969) 241–260.
- Lin, C.T., The computation of probabilities which involve spacings, with applications to the scan statistic, Ph.D. Dissertation (Dept. of Statistics, Florida State University, Tallahassee, FL, 1993).
- Loader, C.R., Large-deviation approximations to the distribution of scan statistics, *Adv. Appl. Prob.* **23** (1991) 751–771.
- Marshall, A.W., and I. Olkin, *Inequalities: The Theory of Majorization with Applications to Combinatorics, Probability, Statistics, and Matrix Theory* (Academic Press, New York, 1979).
- Micchelli, C.A., A constructive approach to Kergin Interpolation in  $\mathcal{R}^k$ : Multivariate B-splines and Lagrange interpolation, *Rocky Mountain J. Math.* **10** (1980) 485–497.
- Naus, J.I., The distribution of the size of the maximum cluster of points on a line, *J. Amer. Statist. Assoc.* **60** (1965) 532–538.
- Naus, J.I., Some probabilities, expectations and variances for the size of the largest clusters and smallest intervals, *J. Amer. Statist. Assoc.* **61** (1966) 1191–1199.
- Neff, N.D., and J.I. Naus, The distribution of the size of the maximum cluster of points on a line, *IMS Series of Selected Tables in Mathematical Statistics* **6** (American Mathematical Society, Providence, RI, 1980).
- Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes – The Art of Scientific Computing*, (Cambridge University Press, Cambridge, 1986).
- Roos, M., Compound Poisson approximations for the number of extreme spacings, *Adv. Appl. Prob.* **25** (1993) 847–874.