TITLE : CHOICE OF BINARY MATRICES UNDER INEQUALITY
RESTRICTIONS ON ROW TOTALS AND COLUMN TOTALS

STUDENT : ANUP KUMAR DE

SUPERVISOR : DR. BIMAL KUMAR ROY (CSU)

INTRODUCTION

Let us consider a population with N = rc units represented by a r x c two-way array. Suppose each row represents a group with respect to some characteristic, and each column represents a group with respect to some other characteristic. Then a practical sampling problem may be
n units, so that from any grou
Samples including more than k
non-preferred, following Sengu
$\pi_{ij} < \pi_i \pi_j \; \forall$ i,j in order to ensure the positiveness of the estimator of variance of the HTE, where π_i and π_{ij} are the first and second order inclusion probabilities respectively. Now if the inclusion probabilities are made constant, then the demand will be automatically satisfied as in the case of SRSWOR. Moreover, data analysis will be extremely simple. This is a problem of deep stratification. Sengupta [1] has posed and partly desolved this problem with k = 2, In that paper he has shown the procedure of sampling in the following cases :

(i) $r = c =$ even , $n \leq r + 2$

(ii) $r = c =$ odd, $n \leq r + 1$

(iii) $r \neq c$, min (r,c) = odd, $n \leq$ min (r,c) + 1

(iv) $r \neq c$, min (r,c) = even, $n \leq$ min (r,c).

Contd.....

In this paper the problem has been solved when r = c for general k, and the sampling schemes along with their implementation as a software package has been shown for the following three cases :

(a) $r = c = 0 \pmod k$, $n \leq r(k-1) + k$

(b) $r = c = 1 \pmod k$, $n \leq (r-1)(k-1)+k$

(c) $r = c = j \pmod k$, $2 \leq j \leq k-1$, $n \leq (r-j)(k-1)+k$

clearly (i) and (ii) above are particular cases, putting k = 2, of (a) and (b) respectively. Case (c) is entirely new, since this case arises only if $k \geq 3$.

In section 1 sampling designs of cases (a) and (b) are described in details. In section 2 design of case (c) is discussed. Section 3 describes the algorithm to implement the design. Section 4 gives some results of the algorithm, as obtained from the computer output. Section 5 discuss about the possible scopes of improvement.

Section 1 :

In this section sampling scheme of the first two cases are described. But before that, some notations and a few lemmas are necessary to be explained.

Let $A_{p,q}\ (1^{c_1}, 2^{c_2}, \ldots, k^{c_k})$ = Number of ways a p x q binary matrix can be filled up by p_k 1s, taking k 1s from each row, in such a way that c_1 columns are filled up by one 1, c_2 columns are filled up by two 1s and so on upto c_k columns are filled up by k 1s. Obviously for a solution to exist, $\sum_{i=1}^{k} c_i = q$ and $\sum_{i=1}^{k} ic_i = p_k$. Hence q can be safely dropped from the notation. Also if some c_i is zero, that term, if it is not the last term, may be optionally omitted. With this notation the following two lemmas can be easily verified.

Lemma 1 : $A_1\ (1^k, 2^0, \ldots, k^0) = 1$

Lemma 2 : $A_p\ (k^p) = \binom{p}{k} A_{p-1}\ ((k-1)^k, k^{p-k})$.

Next observe that the sampling problem is equivalent to the following matrix problem :

Let there be a r x c binary matrix. Fill up the matrix by n 1s so that at most k 1s are put in any row and in any column. Here we are interested about the cases when r = c. So from now on population size, N, will be a perfect square, r^2.

Contd.....

Let f_{ij} denote the number of samples containing both the i-th and j-th population unit. Then in this case, i.e. the case where the matrix is square, it is clear that there are two types of f_{ij} , and hence two types of π_{ij}, depending on whether i-th and j-th unit belong to the same row or same column, or not. Let $a_{r,k} = f_{ij}$ if i-th and j-th unit belong to same row or same column <u>in the r x r array arrangement</u>. [From now on, the underlined phrase will be understood.]

Now let an m x m matrix be filled up with km 1s. Then the total number of ways by which this can be done is $A_m(k^m)$. Then it is easy to verify.

Lemma 3 : $a_{m,k} = \binom{m-2}{k-2} A_{m-1}((k-1)^k, k^{m-k})$.

Let π_{ij} denote the second order inclusion probabilities if i-th and j-th unit belong to same row or same column and π'_{ij} denote the same if otherwise. This convention of π_{ij} and π'_{ij} is followed throughout this paper and whenever there will be more than one design in consideration, they will be written as $\pi_{ij}(D)$ and $\pi'_{ij}(D)$ to correspond the design D. Then the following lemma is true.

Lemma 4 : If (i) n is the sample size,

(ii) i is a fixed population unit,

Contd...

(iii) there are x js such that $j \neq i$ and i,j belong to the same row or same column,

(iv) there are y js such that $j \neq i$ and j is not considered in condition (iii), then

$$x\pi_{ij} + y\pi'_{ij} = (n-1)\pi_i.$$

Next let $N = m^2$, $n = km$ and the sampling design D' is to fill up the m x m matrix by n 1s such that each row and each column contains exactly k 1s. In this case, as well as in all the sampling schemes to be discussed later, first order inclusion probabilities are constant due to the random choice of units. Hence in all these cases π_i = inclusion probabilities of the i-th unit = n/N.

By lemma 2 and lemma 3

$$\pi_{ij}(D') = a_{m,k} / A_m(k^m) = k(k-1) / [m(m-1)].$$

Now by lemma 4,

$$2(m-1)\,\pi_{ij}(D') + (m-1)^2\,\pi'_{ij}(D') = (n-1)\,\pi_i(D')$$

$$\text{or } (m-1)^2\,\pi'_{ij}(D') = (km-1)\,k/m - 2(m-1)\,k\,(k-1)/[m\,(m-1)]$$

$$= k(km-2k+1)/m$$

$$\therefore\ \pi'_{ij}(D') = k(km-2k+1)/[m(m-1)^2].$$

We are now in a position to describe the sampling schemes.

Contd.....

Case (a) : $r = c = 0 \pmod{k}$, $n \leq r(k-1) + k$, $N = r^2$.

Let $r = c = \frac{k(m-1)}{(k-1)}$. Then $n \leq km$.

W.l.g. assume $n = km$, since if $n < km$, then we can further sample n units from km units using SRSWOR.

Sampling scheme (D_1) : From r x r matrix choose an m x m submatrix at random. Then fill up that submatrix by n 1s at random so that each row and each column contains k 1s.

$$\text{Then } \pi_{ij}(D_1) = \frac{\binom{r-1}{m-1}\binom{r-2}{m-2}}{\binom{r}{m}^2} \pi_{ij}(D')$$

$$= \frac{m^2(m-1)}{r^2(r-1)} \cdot \frac{k(k-1)}{m(m-1)} \text{ [by lemma 2 and 3]}$$

$$= \frac{mk(k-1)}{r^2(r-1)}$$

$$\pi'_{ij}(D_1) = \frac{\binom{r-2}{m-2}^2}{\binom{r}{m}^2} \cdot \pi'_{ij}(D') = \frac{m^2(m-1)^2}{r^2(r-1)^2} \cdot \frac{k(km-2k+1)}{m(m-1)^2}$$

$$= \frac{km(km-2k+1)}{r^2(r-1)^2} = \frac{km(kr-r-k+1)}{r^2(r-1)^2}$$

$$= \frac{km(k-1)(r-1)}{r^2(r-1)^2} = \frac{mk(k-1)}{r^2(r-1)}$$

So $\pi_{ij}(D_1) = \pi'_{ij}(D_1)$ = constant, as our requirement.

Contd.....

Case (b) : $r = c = 1 \pmod{k}$, $n \leq (r-1)(k-1) + k$, $N = r^2$.

Let $r = c = \frac{k(m-1)}{k-1} + 1$. Then $n \leq km$.

As in case (a), assume $n = km$.

Sampling scheme (D_2) : Choose design D_{21} with probability f and design D_{22} with probability 1-f, notationally

$$D_2 = f\ D_{21} + (1-f)\ D_{22},$$

where D_{21} is same as D_1 in case (a) and D_{22} is as follows :

Fill up the r x r matrix by (n-1) 1s in such a way that each row and each column contains (k-1) 1s. Then put another 1 in any of the remaining positions.

To show that this design satisfies our demand, we may find $\pi_{ij}(D_{21})$, $\pi'_{ij}(D_{21})$, $\pi_{ij}(D_{22})$, $\pi'_{ij}(D_{22})$ and then find f such that $f\pi_{ij}(D_{21}) + (1-f)\pi_{ij}(D_{22}) = f\pi'_{ij}(D_{21}) + (1-f)\pi'_{ij}(D_{22})$, and $0 \leq f \leq 1$. But $\pi'_{ij}(D_{21})$ and $\pi'_{ij}(D_{22})$ are comparatively difficult to find explicitly. Instead we shall solve a simpler equation exploiting lemma 4. We shall solve the equation,

$$f\pi_{ij}(D_{21}) + (1-f)\pi_{ij}(D_{22}) = n(n-1)/[r^2(r^2-1)] \text{ ---- } (*)$$

In appendix it will be shown that these two equations in f are equivalent.

Contd.....

From case (a) we get $\pi_{ij}(D_{21}) = mk(k-1)/[r^2(r-1)]$. For k=2, the design D_{22} becomes, choose a permutation matrix and then in the rest of the matrix choose another unit.

$\therefore$ Total number of solution = $(2m-1)!\ (2m-1)(2m-2)$,

since $k = 2 \Rightarrow r = c = 2m-1$ and $n = 2m$.

If two elements in the same row are fixed, then one of them must be the specially chosen one. If we delete that then we have a single element in that row and the rest of the matrix can be filled up in $(2m-2)!$ ways to give a permutation matrix.

$$\therefore \pi_{ij}(D_{22}) = 2(2m-2)!\ /[(2m-1)!\ (2m-1)(2m-2)]$$
$$= 1/[(2m-1)^2(m-1)]$$

Now (*) $\Rightarrow f = [n(n-1)/r^2 - (r^2-1)\ \pi_{ij}(D_{22})]/[(r^2-1)(\pi_{ij}(D_{21}) - \pi_{ij}(D_{22}))]$

Here, $\pi_{ij}(D_{21}) - \pi_{ij}(D_{22}) = \dfrac{2m}{(2m-1)^2(2m-2)} - \dfrac{1}{(2m-1)^2(m-1)}$

$$= 1/(2m-1)^2$$

$n(n-1)/r^2 = 2m(2m-1)/(2m-1)^2 = 2m/(2m-1)$.

Contd.....

Correction :

Line 9 of page 9 should be read as :

. . . can be filled up in $(r-2)^{(k-3)}$ ways, then the rest of the matrix can be filled in $A^{(k-1)}_{r-1}((k-2), (k-1)^{r-k+1})$ ways, and lastly . . .

$$\therefore\ f = \left[\frac{2m}{2m-1} - \frac{4m}{(2m-1)^2}\right] \Big/ \frac{4m(m-1)}{(2m-1)^2}$$

$$= \frac{2m(2m-3)}{(2m-1)^2} \Big/ \frac{4m(m-1)}{(2m-1)^2} = \frac{2m-3}{2(m-1)} = 1 - \frac{1}{2m-2} = 1 - \frac{n}{N-1}$$

which agrees with Sengupta [1].

Next to find $\pi_{ij}(D_{22})$ for $k \geq 3$.

Total number of solution = $A_r\ ((k-1)^r)(r^2-(k-1)r)$.

If two elements, say (1,1)-th and (1,2)-th are fixed then either one of them is the specially chosen element or not. If none of them is the specially chosen element then the first row can be filled up in $A_{r-1}((k-2)^{k-1},\ (k-1)^{r-k+1})$ ways, and lastly the special element can be chosen in $(r^2-(k-1)r)$ ways. If one of the fixed two elements is the specially chosen element then suppress that, fill up the rest of the first row in $\binom{r-2}{k-2}$ ways and fill up the rest of the matrix in $A_{r-1}((k-2)^{k-1},(k-1)^{r-k+1})$ ways.

$$\therefore \pi_{ij}(D_{22}) = \left[\binom{r-2}{k-3}A_{r-1}((k-2)^{k-1},\ (k-1)^{r-k+1})(r^2-(k-1)r) + 2\binom{r-2}{k-2}A_{r-1}((k-2)^{k-1},(k-1)^{r-k+1})\right] \Big/ \left[A_r((k-1)^r)(r^2-(k-1)r)\right]$$

Contd.....

$$= \frac{[\binom{r-2}{k-3}(r^2-(k-1)r)+2\binom{r-2}{k-2}]A_{r-1}((k-2)^{k-1},(k-1)^{r-k+1})}{\binom{r}{k-1}A_{r-1}((k-2)^{k-1},(k-1)^{r-k+1})(r^2-(k-1)r)}$$

$$= \frac{(r-2)!}{(k-3)!(r-k)!}\left(r+\frac{2}{k-2}\right) / \frac{r!r(r-k+1)}{(k-1)!(r-k+1)!}$$

$$= (k-1)(r-k+1)(rk-2r+2)/[r(r-1)r(r-k+1)]$$

$$= (k-1)(rk-2r+2)/[r^2(r-1)]$$

$$\therefore \pi_{ij}(D_{21}) - \pi_{ij}(D_{22}) = \frac{n(k-1)}{r^2(r-1)} - \frac{(k-1)(rk-2r+2)}{r^2(r-1)}$$

$$= \frac{(k-1)[n-(rk-2r+2)]}{r^2(r-1)}$$

$$\therefore \text{ Numerator of } f = \frac{n(n-1)}{r^2} - \frac{(r^2-1)(k-1)(rk-2r+2)}{r^2(r-1)}$$

$$= [n(n-1)-(r+1)(k-1)(rk-2r+2)]/r^2$$

$$\therefore f = \frac{n(n-1) - (r+1)(k-1)(rk-2r+2)}{(r+1)(k-1)(n-rk+2r-2)}$$

Contd.....

$$= \frac{nr(k-1)-(r+1)(k-1)(rk-2r+2)}{(r+1)(k-1)(n-rk+2r-2)} \quad [\because\ r = (n-1)/(k-1)]$$

$$= [nr-(r+1)(n-r+1)]/[(r+1)(n-n+r-1)]$$

$$= (r^2-n-1)/(r^2-1)$$

$$= 1-n/(N-1)$$

$\because\ N > 1 \quad \therefore\ f \leq 1.$

Also, $n \leq (N-1) \Rightarrow f \geq 0$

$\therefore\ 0 \leq f \leq 1.$

So finally D_2 becomes

$$D_2 = \left(1 - \frac{n}{N-1}\right) D_{21} + \frac{n}{N-1} D_{22}\ .$$

Section 2 :

In this section sampling scheme of the third case is described.

Case (c) : $r = c = j \pmod k$, $2 \leq j \leq k-1$, $n \leq (r-j)(k-1)+k$.

Let $r = c = \frac{k(m-1)}{k-1} + j$. Then $n \leq km$.

As in case (a), assume $n = km$.

Contd.....

Sampling scheme (D_3) : Choose design D_{31} with probability f and design D_{32} with probability (1-f), notationally,

$$D_3 = f\ D_{31} + (1-f)\ D_{32}\ ,$$

where D_{31} is same as D_1 in case (a) and D_{32} is as follows :

Fill up the r x r matrix by mk + (jk - k - j) 1s in such a way that each row and each column contain exactly (k-1) 1s. Then remove any (jk - k - j) 1s.

To show that this design satisfies our demand, we shall solve $f\pi_{ij}(D_{31}) + (1-f)\pi_{ij}\ (D_{32}) = \frac{n(n-1)}{r^2(r^2-1)}$, due to the reason discussed in section 1.

Now, $\pi_{ij}\ (D_{31}) = \frac{n(k-1)}{r^2(r\ -1)}$ as before.

To find $\pi_{ij}(D_{32})$, note that

total number of solutions = $A_r\ ((k-1)^r)\binom{(k-1)r}{jk-k-j}$

$$= \binom{r}{k-1}\ A_{r-1}\ ((k-2)^{k-1},\ (k-1)^{r-k+1})\ \binom{(k-1)r}{jk-k-j}.$$

Contd.....

Following results are used for simplification :

$$r = \frac{k(m-1)}{k-1} + j = \frac{km-k}{k-1} + j = \frac{n-k}{k-1} + j \quad [\because n = mk]$$

$$\therefore \quad n = (k-1)(r-j)+k = kr-r-jk + j+k \text{ and}$$

$$n-1 = (k-1)(r-j) + (k-1) = (k-1)(r-j+1).$$

Now we calculate $\pi_{ij}(D_{32})$.

If two elements in the first row are set to 1, then the rest of the first row can be filled up in $\binom{r-2}{k-3}$ ways, then the remaining rows can be filled up in $A_{r-1}((k-2)^{k-1}, (k-1)^{r-k+1})$ ways and then $(jk-k-j)$ units can be cancelled in $\binom{(k-1)r-2}{jk-k-j}$ ways.

$$\therefore \pi_{ij}(D_{32}) = \frac{\binom{r-2}{k-3} A_{r-1}((k-2)^{k-1}, (k-1)^{r-k+1})\binom{(k-1)r-2}{jk-k-j}}{\binom{r}{k-1}A_{r-1}((k-2)^{k-1},(k-1)^{r-k+1})\binom{(k-1)r}{jk-k-j}}$$

$$= \frac{(r-2)!(k-1)!(r-k+1)!}{(k-3)!(r-k+1)!r!} \cdot \frac{((k-1)r-2)!(jk-k-j)!((k-1)r-jk+k+j)}{(jk-k-j)!((k-1)r-jk+k+j-2)!((k-1)r)!}$$

$$= \frac{(k-1)(k-2)}{r(r-1)} \cdot \frac{[(k-1)r-jk+k+j][(k-1)r-jk+k+j-1]}{(k-1)r((k-1)r-1)}$$

$$= \frac{(k-2)(kr-r-jk+k+j)[(k-1)r-(j-1)(k-1)]}{r^2(r-1)(kr-r-1)}$$

Contd.....

$$= \frac{(k-2)(kr-r-jk+k+j)(k-1)(r-j+1)}{r^2(r-1)(kr-r-1)}$$

$$= \frac{(k-2)\ (k-1)(r-j+1)n}{r^2(r-1)(kr-r-1)}$$

$$= \frac{(k-2)(n-1)n}{r^2(r-1)(kr-r-1)}$$

$$\therefore\ f = \frac{\frac{n(n-1)}{r^2} - (r^2-1)\pi_{ij}(D_{32})}{(r^2-1)(\pi_{ij}(D_{31}) - \pi_{ij}(D_{32}))}$$

$$= \frac{n(n-1)/r^2-(r^2-1)n(n-1)(k-2)/[r^2(r-1)(kr-r-1)]}{(r^2-1)[n(k-1)/[r^2(r-1)]-n(k-2)\qquad (n-1)/[r^2(r-1)(kr-r-1)]]}$$

$$= \frac{(n-1)[1-(r+1)(k-2)/(kr-r-1)]}{(r+1)[k-1-(k-2)(n-1)/(kr-r-1)]}$$

$$= \frac{(n-1)[(kr-r-1)-(r+1)(k-2)]/(kr-r-1)}{(r+1)[(k-1)(kr-r-1)-(k-2)(n-1)](kr-r-1)}$$

$$= \frac{(n-1)(kr-r-1-kr+2r-k+2)}{(r+1)[(k-1)(kr-r-1)-(k-2)(k-1)(r-j+1)]}$$

$$= \frac{(k-1)(r-j+1)(r-k+1)}{(k-1)(r+1)[kr-r-1-(kr-jk+k-2r+2j-2)]}$$

$$= (r-j+1)(r-k+1)/[(r+1)(r+jk-k-2j+1)]$$

Contd.....

Next to show that $0 \leq f \leq 1$.

$r \geq j$ (by definition of r) $\Rightarrow r - j + 1 > 0$

Also, $r + 1 > k$ $\because k > r$ is meaningless.

$\therefore r - k + 1 > 0$.

Next $r + 1 > 0$.

Finally $r + jk - k - 2j + 1 = r - j + jk - k - j + 1$

$= (r-j) + (j-1)(k-1)$.

But $r \geq j$, $j \geq 2$, $k \geq 3 \Rightarrow r + jk - k - 2j + 1 > 0$.

$\therefore f > 0$.

To show $f \leq 1$, we have to show

$(r - j + 1)(r-k+1) \leq (r+1)(r+jk-k-2j+1)$

i.e. $r^2 - rk + r - jr + jk + j + r - k + 1$

$\leq r^2 + rjk - rk - 2rj + r + r + jk - k - 2j + 1$

i.e. $j \leq rjk - rj - 2j$

i.e. $1 \leq rk - r - 2$

i.e. $r + 3 \leq rk$ which is true since $r \geq k \geq 3$.

So finally D_3 becomes

$$D_3 = \frac{(r-j+1)(r-k+1)}{(r+1)(r+jk-k-2j+1)} D_{31} + \left(1 - \frac{(r-j+1)(r-k+1)}{(r+1)(r+jk-k-2j+1)}\right) D_{32} .$$

Contd.....

Section 3 :

In this section a brief description of the algorithm is given. To understand this algorithm no knowledge of high level programming language is required. The interested reader is referred to the source program (name is SAMPLE) of this algorithm, available with the author. .

Description of the algorithm :

Before describing the algorithm, a subalgorithm GENERATE is necessary to be described.

GENERATE : This algorithm inputs a matrix, say, S, and integers k and L. S is a binary matrix and it is filled up by Os and 1s. If S is filled up in such a way that the row sums and column sums are all k then we call it a solution. This subalgorithm can generate all such solutions in a systematic way. But, according to our need, it generates only upto the L-th solution.

Now the main algorithm is described :

Step 1 : Input population size (N), Sample size (n) and maximum number (k) of units allowed from any group.

Step 2 : Calculate (a) $r = \sqrt{N}$ [N should be a squared number]

(b) $j = r \pmod k$

(c) $m = [r/k].(k-1)+1.$

Contd,....

Step 3 : If j = 0 then perform Step 4,
if j = 1 then perform Step 5,
if otherwise then perform Step 6.

Step 4 : From r rows and r columns choose m rows and m columns at random. Find in how many ways an mxm binary matrix can be filled up by mk 1s so that each row and column sum become k. To find this number, say T, a recursive relation is used, the relation being discussed in detail at the end of this section. Next, choose a random number, say L, between 1 and T. Then using GENERATE get the L-th solution. Now perform Step 8 to construct the sampling units. [This corresponds to design D_1.]

Step 5 : Calculate f = 1 - n/(N-1). Perform Step 7.

Step 6 : Calculate $f = [(r-j+1)(r-k+1)] / [(r+1)(r+jk-k-2j+1)]$. Perform Step 7.

Step 7 : Choose a random number $s \in [0,1]$. If $s < f$ then perform Step 4. [This corresponds to design D_{21} or D_{31} depending on the previous step.]
Otherwise find in how many ways an r x r binary matrix can be filled up by r(k-1) 1s so that each row and column sum become k-1. As in Step 4, get a random solution of this form using GENERATE. [This corresponds to design D_{22} or D_{32} depending on the previous step.]

Contd.....

Perform Step 9 if the previous step is Step 5, otherwise perform Step 10.

Step 8 : At this stage we have an m x m submatrix filled up with mk 1s. To interpret 1s of the submatrix as sampling units, note that (k,ℓ)-th entry of the main r x r matrix corresponds to [(k-1)r+ℓ]-th sampling unit. So if (i,j)-th entry of the submatrix corresponds to (k,ℓ)-th entry of the matrix, then output [(k-1)r+ℓ] if (i,j)-th entry of the submatrix is 1. In this way, output all the sampling units and terminate the algorithm.

Step 9 : At this stage, we have an r x r matrix filled up with another (mk-1) 1s. Make/zero entry 1 at random. Then if (k,ℓ)-th entry of the matrix is 1, output [(k-1)r+ℓ]. In this way, output all the sampling units and terminate the algorithm.

Step 10: At this stage, we have an r x r matrix filled up with mk+(jk-k-j) 1s. Choose any (jk-k-j) of those 1s at random and make them 0. Then if (k,ℓ)-th entry of the matrix is 1, output [(k-1)r+ℓ]. In this way, output all the sampling units and terminate the algorithm.

Contd.....

Note that in the algorithm it is assumed that $n = mk$. But if $n < mk$, then n units from the mk chosen units may be selected at random for the required output.

To find the number of ways an m x m binary matrix can be filled up by mk 1s so that each row and column sum become k, we use the following recursive relation :

$$\text{Result : } A_m\,(1^{c_1}, 2^{c_2}, \ldots, k^{c_k}) = \sum_{I} \binom{c_1}{j_1} \ldots \binom{c_k}{j_k} A_{m-1}(1^{c_1-j_1}, \ldots, k^{c_k-j_k})$$

where $I = [(j_1, \ldots, j_k) : 0 \le j_i \le c_i \ \forall_i$, and $\sum_{i=1}^{k} j_i = k\ ]$ with the convention that $\binom{n}{r} = 0$ if $r > n$.

Initial condition, (as given in lemma 1), is

$$A_1(1^k, 2^0, \ldots, k^0) = 1.$$

Note that the set I is difficult to be implemented. It is implemented with the help of a recursive algorithm. Hence to evaluate the total number of solutions we need two algorithms which are both self recursive as well as mutually recursive. The detail of these algorithms are not discussed here, since they may be found in the source program.

Contd.....

Finally we should note that it is not a good idea to attempt to find the recursive formula for each k separately. Because even for k = 3, the right hand side will contain ten terms. Even if one is very patient to find explicit formula for, say, upto k = 10, then also that program will not work for k > 10. Hence it is necessary to implement the set I by a recursive algorithm.

Section 4 :

In this section some sample output of the algorithm are given. In each case we have taken the sample size to be mk. So no further subsample is required. Notations used in the results are :

N : population size, assume population units to be labelled 1,2,....,N,

n : sample size,

k : maximum allowed number of units from any group to be sampled,

D : the design chosen for sampling; available designs are D_1, D_{21}, D_{22}, D_{31}, D_{32} as indicated in this paper,

S : the sample set,

f : the convex combination factor of D_2 or D_3 as the case may be, as indicated in this paper.

Contd.....

Output 1 : N = 16, n = 6, k = 2, D = D_1 ,

S = [6, 7, 11, 12, 14, 16]

Output 2 : N = 25, n = 6, k = 2, D = D_{21} , f = 0.7500

S = [2, 4, 9, 10, 22, 25]

Output 3 : N = 25, n = 9, k = 3, D = D_{32} , f = 0,4000

S = [1, 3, 8, 10, 12, 14, 17, 19, 25]

Output 4 : N = 36, n = 8, k = 2, D = D_1 ,

S = [2, 4, 15, 16, 20, 24, 33, 36]

Output 5 : N = 36, n = 15, k = 3, D = D_1

S=[2,4,5,15,17,18,22,23,24,26,27,28,32,33,36]

Output 6 : N = 49, n = 15, k = 3, D = D_{21} , f = 0.6875

S = [9,12,13,15,19,20,23,25,26,29,32,34,36,37,39]

Output 7 : N = 49, n = 15, k = 3, D = D_{22}, f = 0.6875

S = [5,7,11,13,17,18,27,28,31,33,36,37,43,44,46]

Output 8 : N = 64, n = 15, k = 3, D = D_{31} , f = 0.5833

S = [2,3,6,10,11,14,22,23,24,27,31,32,42,47,48]

Contd.....

Output 9 : N = 64, n = 15, k = 3, D = D_{32} , f = 0.5833

S = [6,8,13,15,22,23,29,32,35,36,43,44,49,50,57]

Output 10: N = 81, n = 21, k = 3, D = D_1

S = [4,7,9,12,15,16,22,25,27,31,33,36,55,56,60,64,65,
66,73,74,75]

Output 11: N = 100, n = 21, k = 3, D = D_{21} , f = 0.7879

S = [7,8,10,14,17,18,46,47,50,66,68,70,72,73,76,
82,83,84,92,93,94]

Output 12: N = 121, n = 21, k = 3, D = D_{32} , f = 0.6818

S = [10,11,20,22,29,31,41,43,51,52,60,61,71,72,
80,81,91,101,111,112]

Output 13: N = 64, n = 28, k = 4, D = D_1

S = [1,6,7,8,10,14,15,16,19,20,23,24,27,30,31,
32,33,34,36,38,41,42,43,44,49,50,51,52]

Output 14: N = 81, n = 28, k = 4, D = D_{21} , f = 0.6500

S = [4,7,8,9,10,16,17,18,20,24,25,27,33,34,35,36,
46,47,49,53,55,56,58,60,73,74,76,78]

Contd.....

Output 15 : N = 81, n = 28, k = 4, D = D_{22} , f = 0.6500

S = [5,8,9,15,16,18,25,26,27,29,31,34,35,40,41,42,
49,50,51,55,56,57,64,65,66,73,74,75]

Output 16 : N = 100, n = 28, k = 4, D = D_{31} , f = 0.5207

S = [16,17,18,19,36,37,38,39,42,47,48,49,63,65,
67,69,72,73,75,78,82,83,85,86,92,93,95,96]

Section 5 :

The results and algorithm discussed in this paper can be improved in one or more ways. We indicate some of these in this section. First the restriction on the sample size should be relaxed. Second, the number of population unit per cell of the matrix may be increased. Third, in scheme (a), the number of representatives of a group in a sample is either 0 or k. So for k = 7, say, any sample will contain either 7 elements or none from a group. For n < mk, this discontinuation may disappear partly or fully. In scheme (b) this number of representatives is 0 or k-1 or k. Hence it is discontinuous for $k \geq 3$ and n = mk. In scheme (c) if D_{31} is chosen then the number of representatives is 0 or k. If D_{32} is chosen, then the problem depends on j. If j = 2, then jk-k-j=2k-k-2 = k-2, so number of representatives lies between 1 and k-1. For $j \geq 3$, the value of jk-k-j increases.

Contd.....

Hence in scheme (c) the discontinuity disappears completely. Fourth the algorithm takes most of its time to execute the subalgorithm GENERATE. To get the L-th solution matrix, this subalgorithm generates all the previous (L-1) solutions, which ideally it should not. We think that the recursive relation

$$A_m(1^{c_1},\ldots,k^{c_k}) = \sum_I \binom{c_1}{j_1} \ldots \binom{c_k}{j_k} A_{m-1}(1^{c_1-j_1},\ldots,k^{c_k-j_k})$$

may be exploited to directly generate the L-th solution.

Acknowledgement :

I would like to thank Dr. B.K.Sinha, Professor, Indian Statistical Institute, Calcutta, for exposing the problem and also for his valuable advice.

Reference :

[1] S.Sengupta, Construction of some noninvariant balanced sampling designs, CSA bulletin, Vol. 31, Sept. and Dec. 1982.

APPENDIX

In section 1, two equations in f are claimed to be equivalent. Here we shall show a little more.

Let $D = fD_1 + (1-f)\ D_2$, where D, D_1, D_2 are sampling designs, and f is to be chosen in such a way that $\pi_{ij}(D) = \pi'_{ij}(D)$.

Let $x_1 = \pi_{ij}(D_1)$, $x_2 = \pi'_{ij}(D_1)$, $y_1 = \pi_{ij}(D_2)$, $y_2 = \pi'_{ij}(D_2)$, $z_1 = \pi_{ij}(D)$ and $z_2 = \pi'_{ij}(D)$.

From lemma 4, we get

$$2(r-1)x_1 + (r-1)^2\ x_2 = n(n-1)/r^2 \quad \ldots\ldots\ldots \text{(i)}$$

where $r^2 = N$ is the population size and n is the sample size as throughout this paper.

(i) $\Rightarrow x_2 = [c - 2(r-1)x_1]/(r-1)^2$ where $c = n(n-1)/r^2$.

Also, $2(r-1)y_1 + (r-1)^2 y_2 = n(n-1)/r^2$ .....(ii)

$\Rightarrow y_2 = [c-2(r-1)y_1]/(r-1)^2$.

Next, by definition $z_1 = fx_1 + (1-f)y_1$ .....(iii)

and $z_2 = fx_2 + (1-f)y_2$ .....(iv). Our aim is to find f in such a way that $z_1 = z_2$.

Contd.....

Note that since lemma 4 is applicable for z_1 and z_2 also, so $2(r-1)z_1 + (r-1)^2 z_2 = n(n-1)/r^2$ ..... (v).

We now argue that if an f exists such that $z_1 = z_2$ then (v) implies $z_1 = z_2 = c/(r^2-1)$. So we could solve the equation

$$fx_1 + (1-f)y_1 = c/(r^2-1) \text{ .....(vi) or,}$$

$$fx_2 + (1-f)y_2 = c/(r^2-1) \text{ .....(vii)}$$

But a straight cut way to find f is to solve

$fx_1 + (1-f)y_1 = fx_2 + (1-f)y_2$ .....(viii). We shall now show that (vi),(vii) and (viii) are equivalent.

If we solve (viii), we get

$$f = (y_2 - y_1) / (x_1 - y_1 - x_2 + y_2).$$

Putting the value of y_2 in terms of y_1 we get,

$$y_2 - y_1 = \frac{c-2(r-1)y_1 - (r-1)^2 y_1}{(r-1)} = \frac{c-(r^2-1)y_1}{(r-1)^2}$$

Similarly,

$$x_2 - x_1 = \frac{c-(r^2-1)x_1}{(r-1)^2}$$

$$\therefore (x_1 - x_2) + (y_2 - y_1) = \frac{(r^2-1)(x_1-y_1)}{(r-1)^2}$$

$$\therefore f = \frac{c-(r^2-1)y_1}{(r-1)^2} / \frac{(r^2-1)(x_1-y_1)}{(r-1)^2} = \frac{c-(r^2-1)y_1}{(r^2-1)(x_1-y_1)}$$

Contd.....

If we solve (vi) we get,

$$f(x_1-y_1) = c/(r^2-1) - y_1$$

$$\therefore f = \frac{c-(r^2-1)y_1}{(r^2-1)(x_1-y_1)}$$

So (vi) and (viii) give the same solution.

Next, if we solve (viii), we get

$$f = \frac{c-(r^2-1)y_2}{(r^2-1)(x_2-y_2)}$$

So finally we shall show,

$$\frac{c-(r^2-1)y_1}{x_1-y_1} = \frac{c-(r^2-1)y_2}{x_2-y_2}$$

Now, $x_2y_1 = \frac{c-(2r-2)x_1}{(r-1)^2} y_1$ and

$$y_2x_1 = \frac{c-(2r-2)y_1}{(r-1)^2} x_1$$

$$\therefore x_2y_1 - x_1y_2 = \frac{cy_1-(2r-2)x_1y_1-cx_1+(2x-2)x_1y_1}{(r-1)^2}$$

$$= \frac{c(y_1-x_1)}{(r-1)^2} \quad .....(ix)$$

Contd.....

Next, (i) - (ii) $\Rightarrow 2(r-1)(x_1-y_1) + (r-1)^2(x_2-y_2) = 0$

or $2(x_1 - y_1) = -(r-1)(x_2-y_2)$

$$\therefore\ x_2 - y_2 = -2(x_1-y_1)/(r-1)$$

$$\therefore\ \frac{c(x_2-y_2-x_1+y_1)}{r^2-1} = \frac{c}{r^2-1}\ [\ (x_2 - y_2) - (x_1 - y_1)\ ]$$

$$= \frac{c}{r^2-1} \cdot [-(x_1-y_1)\ (\ \frac{2}{r-1} + 1)] = c(y_1-x_1)/(r-1)^2 \ldots\ldots\ldots(x)$$

Finally, (ix) and (x) imply,

$$c(x_2 - y_2 - x_1 + y_1\ ) = (r^2 - 1)\ (x_2y_1 - x_1y_2)$$

$$= (r^2 - 1)\ (x_2y_1 - y_1y_2 - x_1y_2 + y_1y_2\ )$$

or, $\quad c(x_2-y_2) - (r^2-1)y_1(x_2-y_2) = c(x_1-y_1) - (r^2-1)y_2(x_1-y_1)$

or, $\quad \frac{c-(r^2-1)y_1}{x_1-y_1} = \frac{c-(r^2-1)y_2}{x_2-y_2}$, proved.

Source Program Listing :
Dissertation Series.
By ANUP KUMAR DE

T102
30/8/88

```
{$u+}
program SAMPLING;

type
     settype = set of 1..100;
     vector = array [1..10] of byte;
     matrix = array [1..10,1..10] of byte;
     ScumV = record  {Set cum Vector}
             case class : 1..2 of
             1 : (content : vector);
             2 : (contentset : settype)
             end; {of record}

var
    Outfile : text;
    FileName : string[12];
    PopulationSize,SampleSize,K : integer;
    n,R,Remainder,M : integer;
    Pattern : 1..3;

{N = PopulationSize: Population units are labelled 1,2,...,N.
 R : Population units are arranged in a R X R array. Hence R = sqrt(N).
 K : At most K units are allowed to be sampled from any group. So samples
     containing more than K units from a particular group are nonpreferred.
 M,Remainder : They are such that R = K * (M - 1) / (K - 1) + Remainder.   }

{-------------------------------------------------------------------------}
{ - - - - - - - - - - - - procedure ScreenDesign- - - - - - - - - - - - - -}
{-------------------------------------------------------------------------}

procedure ScreenDesign;
var
    x,y : integer;
    procedure TwoChar (x1,y1,x2,y2 : integer);
    begin
         gotoxy (x1,y1);
         write( chr(176) );
         gotoxy (x2,y2);
         write( chr(176) )
     end;

begin
     clrscr;
     for x  := 5 to 75 do
         TwoChar (x,5,x,20);
for y  := 5 to 20 do
     TwoChar (5,y,75,y);
window (6,6,74,19);
gotoxy (1,1);
writeln;
LowVideo;
writeln ('                    A sampling scheme based on ');
writeln;
HighVideo;
writeln('          ''Choice on binary matrices under inequality'');
writeln('           restrictions on row totals and column totals.'');
LowVideo;
writeln;
writeln('                    A dissertation work');
writeln;'                          by');
writeln('                    ANUP KUMAR DE ');
HighVideo;
writeln('                    guided by');
LowVideo;
writeln;
writeln('
```

```
      HighVideo;
      writeln('                              Dr. BIMAL KUMAR ROY');
      LowVideo;
      delay (10000);
      window (1,1,80,24)
end; (of ScreenDesign)

{-------------------------------------------------------------------------------}
{ - - - - - - - - - - - - procedure InputParameters - - - - - - - - - - - - - -}
{-------------------------------------------------------------------------------}

{This procedure either inputs or constructs the required parameters. }

procedure InputParameters;
var
   Temporary : integer;
begin
      write('Population size (should be a squared number) =? ');
      readln(PopulationSize);
      R := trunc( sqrt(PopulationSize) );
      if PopulationSize <> (R * R)
      then begin
                HighVideo;
                writeln('Your population size is not a squared number. So the program is halted.');
                writeln('Press R to execute the program again');
                HALT
           end;
      write('Sample size =? ');
      readln(SampleSize);
      write('Maximum # allowed units from any group =? ');
      readln(K);

      writeln(Outfile);
      writeln(Outfile);
      writeln(Outfile,'Given the following data :');
      writeln(Outfile);
      writeln(Outfile,'Population size = ',PopulationSize:3);
      writeln(Outfile,'Sample size     = ',SampleSize:3);
      writeln(Outfile,'Samples including more than',K : 3,
                      ' units from any group are nonpreferred.');
      writeln(Outfile,'-------------------------------------------------------------------------------');

      writeln(Outfile);
      writeln(Outfile,'Hence the calculated parameters are :');
      writeln(Outfile);
      write(Outfile,'r=',R : 2);
      Remainder := R mod K;
      write(Outfile,'   j=',Remainder : 2);
      Temporary := R div K;
      M := Temporary * (K - 1) + 1;
      writeln(Outfile,'   m=',M : 2);
      n := M * K;
      writeln(Outfile,'-----------------------------------------');
      writeln(Outfile);
      writeln(Outfile,'At most ',n : 3,' units may be sampled.');
      if SampleSize > n
      then begin
                writeln(Outfile,'So the sample size has been cut down to ',n :3,'.');
                SampleSize := n
           end;
      writeln(Outfile,'-----------------------------------------');
      writeln(Outfile);
      writeln('Press space bar to continue . . .');
      while not keypressed do ;
      clrscr
end; (of InputParameters)

{-------------------------------------------------------------------------------}
{- - - - - - - - - - - - -function NcombinationR - - - - - - - - - - - - - - - }
{-------------------------------------------------------------------------------}

{Purpose of this function is obvious & is indicated by its name.}
```

```
function NcombinationR(N,R : integer) : integer;
{ It is assumed that N and R are nonnegative. }
var
   I,Temporary,Nplus1 : integer;
begin
     if N < R
     then NcombinationR := 0
     else if (R = 0) or (R = N)
          then NcombinationR := 1
          else begin
               if (N - R) < R
               then R := N - R; {Since nCr = nCn-r}
               Temporary := N;
               Nplus1 := N + 1;
               for I := 2 to R do
                   Temporary := Temporary * (Nplus1 - I) div I;
               NcombinationR := Temporary
               end
end; {of NcombinationR}

{-------------------------------------------------------------------------}
{ - - - - - - - - - - - function f - - - - - - - - - - - - - - - - - - - - }
{-------------------------------------------------------------------------}

{This function is active when Remainder <> 0.It returns a value of f, 0 <= f < 1,
depending on whether Remainder is 1 or not. In any case the procedure Choose-
Sample chooses the ordinary scheme with probability f.}

function f : real;
var
   ttt : real;
   Rplus1,Numerator,Denominator : integer;
begin
     case Remainder of
     1 : ttt := 1.0 - n / (PopulationSize - 1.0);
   else  begin
         Rplus1 := R + 1;
         Numerator := (Rplus1 - Remainder) * (Rplus1 - K);
         Denominator := Rplus1 * (Rplus1 - K + Remainder * (K - 2));
         ttt := Numerator / Denominator
         end
     end {case};
     writeln(Outfile,'f=',ttt:6:4);
     f := ttt
end; {of function f}

{-------------------------------------------------------------------------}
{ - - - - - - - - - - - procedure ChooseSet - - - - - - - - - - - - - - - }
{-------------------------------------------------------------------------}

{ This procedure chooses S numbers from the numbers 1,2,...,N and form
  the set LocalSet with those chosen numbers. The selection procedure
  is SRSWOR.}

procedure ChooseSet(N,S : integer; var Units : ScumV);
var
   I,U : integer ;
   LocalSet : settype;
{ I : Counts the number of units chosen so far.
  Units : Represents a number among 1,2,...,N. }

begin {Initially there is no element in LocalSet.}
     I := 0;
     LocalSet := [];
     if Units.class = 2
     then Units.contentset := [];
     repeat
     {random is a built-in function which produces a random number lying
      between 0 to N-1. Hence U is a random number lying between 1 and N.}
          U := random(N) + 1;
          if not (U in LocalSet)
```

```
                    LocalSet := LocalSet + [U];
                    I := I + 1 (One more unit is chosen. )
                end
    until (I = S);

    I:=1;
    for U := 1 to N do
        if U in LocalSet
        then
            if Units.class = 1
            then begin
                Units.content[I] := U;
                I := I + 1
                end
            else Units.contentset := Units.contentset + [U];
    writeln(Outfile)
end; (of ChooseSet)

{------------------------------------------------------------------------------}
{- - - - - - - - - - - - procedure count - - - - - - - - - - - - - - - - - - - -}
{------------------------------------------------------------------------------}

(This procedure counts the total # solutions of the following problem :
 Let there be a binary matrix of dimension NoOfRows X NoOfColumns which is to
 be filled up by NoOfRows X k 1s such that each row will contain k 1s. So
 k : is the maximum # 1s that can be put in any row/column.
 C : is a vector of dimension k, such that C[I] gives the # columns where
     I more 1s can be put. Hence C[1]+C[2]+...+C[k] = NoOfColumns.)

procedure count(NoOfRows,k : integer; C,: vector; var result : real);
var
   Temporary : real;
   B : vector;
   I : integer;

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - procedure count_loop - - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

  ( If we say C[I] represents the # I-th type column, then this procedure
    choses J Index-th type column(s), 0 <= J <= min(Remaining , C[Index]),
    and multiply the coefficient by (c[index] C j). Thus by calling itself
    recursively it generates a single term in the right hand side and calling
    the main procedure mutually recursively it generates the whole right hand
    side of the recursive formula to evaluate total # solutions.)

   procedure count_loop(Index,Remaining,Coefficient : integer);
   var
      J,NewCoefficient,MaximumPossible : integer;
   begin
   if Remaining = 0
   then begin
            for J := 1 to (Index - 1) do
                B[J] := C[J];
            B[Index] := B[Index] + C[Index];
            count(NoOfRows - 1,k,B,result);
            Temporary := Temporary + Coefficient * result
        end
   else if Index = 1
        then if C[1] >= Remaining
            then begin
                    B[1] := B[1] + C[1] - Remaining;
                    NewCoefficient := Coefficient * NcombinationR(C[1],Remaining);
                    count(NoOfRows - 1,k,B,result);
                    Temporary := Temporary + NewCoefficient * result;
                end
            else
        else
        begin
            if Remaining <= C[Index]
            then MaximumPossible := Remaining
```

```
                else MaximumPossible := C[Index];
                for J := 0 to MaximumPossible do
                    begin
                        if Index = k
                        then B[k] := C[k] - J
                        else B[Index] := B[Index] + C[Index] - J;
                        B[Index - 1] := J;
                        NewCoefficient := Coefficient * NcombinationR(C[Index],J);
                        count_loop(Index - 1,Remaining - J,NewCoefficient);
                        if Index <> k
                        then B[Index] := B[Index] + J - C[Index]
                    end
            end
    end; (of count_loop)


{ - - - - - - - - - - - - body of count - - - - - - - - - - - - - - - - - - -}
begin
    if (NoOfRows = 1)
    then if C[1] = k
        then result := 1.0
        else result := 0.0
    else if k = 1
        then begin
            result := 1;
            for I := 2 to NoOfRows do
                result := result * I
            end
        else begin
            Temporary := 0.0; (Temporary will accumulate the values of each term in the right hand side.)
            count_loop(k,k,1); (Coefficient is initialised to 1.)
            result := Temporary
            end
end; (of count)

{-------------------------------------------------------------------------------}
{- - - - - - - - - - - - procedure Construct- - - - - - - - - - - - - - - - - -}
{-------------------------------------------------------------------------------}

procedure Construct(SS : matrix);
var
    BadUnits,LastUnit,I : integer;
    Cancel,Row,Column : ScumV;

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - -Procedure Printmatrix - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

procedure printmatrix (ToBePrinted : matrix; NoOfRows,NoOfColumns : integer);
var
   I,J : integer;
begin
    write(Outfile,'   ');
    for I := 1 to NoOfColumns do
        if Pattern = 1
        then write(Outfile,Column.content[I] : 4)
        else write(Outfile,I : 4);
    writeln(Outfile);
    write(Outfile,'   ');
    for I := 1 to (4 * NoOfColumns + 2) do
        write(Outfile,'-');
    writeln(Outfile);
    for I := 1 to NoOfRows do
        begin
            if Pattern = 1
            then write(Outfile,Row.content[I] : 2,'(')
            else write(Outfile,I:2,'(');
            for J:= 1 to NoOfColumns do
                write(Outfile,ToBePrinted [I,J]:4);
            writeln(Outfile,'  )')
        end;
    write(Outfile,'  ');
```

```
        write(Outfile,'-');
    writeln(Outfile)
end;

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - procedure Construct1 - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

procedure Construct1;
var
   I,J : integer;
begin
    write(Outfile,'  ');
    for I := 1 to (3 * n) do
        write(Outfile,'-');
    writeln(Outfile);
    write(Outfile,' [');
    for I := 1 to M do
        for J := 1 to M do
            if SS[I,J] = 1
            then write(Outfile,(R * (Row.content[I] - 1) + Column.content[J]):3);
    writeln(Outfile,' ]');
    write(Outfile,'  ');
    for I := 1 to (3 * n) do
        write(Outfile,'-');
    writeln(Outfile);
end; (of Construct1)

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - procedure Construct2 - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

procedure Construct2;
var
   I,J,T1,T2,Temp,row,col : integer;
{T1,T2 are temporary variables used for two purposes as indicated below.}

begin
    write(Outfile,'  ');
    for I := 1 to (3 * n + 5) do
        write(Outfile,'-');
    writeln(Outfile);
    write(Outfile,' [');
    for I := 1 to R do
        for J := 1 to R do
            if SS[I,J] = 1
            then write(Outfile,(R * (I - 1) + J) : 3);

    T1 := LastUnit div (R - K + 1);  {Here with the help of T1 & T2, we decide }
    T2 := LastUnit mod (R - K + 1);  {about the row in which we should search. }
    if T2 = 0
    then begin
        row := T1;
        T2 := R - K + 1
        end
    else
        row := T1 + 1;

    T1 := 0;  {Now in the indicated row we have to find the T2-th empty entry.}
    J := 0;   {T1 now counts # empty entries examined. }
    repeat
        J := J + 1;
        if SS[row,J] = 0
        then T1 := T1 + 1
    until (T1 = T2);
    write(Outfile,' and',(R * (row - 1) + J) : 3);
    writeln(Outfile,' ]');
    write(Outfile,'  ');
    for I := 1 to (3 * n + 5) do
        write(Outfile,'-');
```

```
      writeln(Outfile)
end; (of Construct2)

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - procedure Construct3 - - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

procedure Construct3;
var
   I,J,T : integer;
begin
     write(Outfile,'  ');
     for I := 1 to (3 * n) do
         write(Outfile,'-');
     writeln(Outfile);
     write(Outfile,' [');
     T := 0;
     for I := 1 to R do
         for J := 1 to R do
             if SS[I,J] = 1
             then begin
                  T := T + 1;
                  if not (T in Cancel.contentset)
                  then write(Outfile,(R * (I - 1) + J) : 3)
                  end;
     writeln(Outfile,' ]');
     for I := 1 to (3 * n) do
         write(Outfile,'-');
     writeln(Outfile)
end; (of Construct3)

{- - - - - - - - - - - - - - - body of Construct - - - - - - - - - - - - - - - - - - - - - - - - - }
begin
     case Pattern of
     1 : begin
         Row.class := 1;
         ChooseSet(R,M,Row);
         Column.class := 1;
         ChooseSet(R,M,Column);
         printmatrix(SS,M,M);
         writeln(Outfile);
         writeln(Outfile,'The selected sampling units are :');
         Construct1
         end;
     2 : begin
         printmatrix(SS,R,R);
         writeln(Outfile);
         writeln(Outfile,'The selected sampling units are :');
         LastUnit := random( R * (R - (K - 1))) + 1;
         Construct2
         end;
     3 : begin
         printmatrix(SS,R,R);
         writeln(Outfile);
         writeln(Outfile,'The selected sampling units are :');
         BadUnits := Remainder * (K - 1) - K;
         Cancel.class := 2;
         ChooseSet (R * (K - 1),BadUnits,Cancel);
         Construct3
         end
     end; (case)
     writeln('                    * * * G O O D   B Y E * * *');
     close(Outfile);
     HALT
end; (of Construct)

{------------------------------------------------------------------------------}
{ - - - - - - - - - - - - - procedure GenerateMatrix - - - - - - - - - - - - - - }
{------------------------------------------------------------------------------}

(This procedure finds the RequiredNumber-th solution matrix of dimension
 NoOfRows X NoOfColumns, subject to the restriction that no more than k units
```

```
procedure GenerateMatrix (NoOfRows,NoOfColumns,k : integer; RequiredNumber : real);

var
    Resource ,ColumnSum : integer;
    Depth,EmptyColumn   : integer;
    NoOfSolutions : real;
    Columns       .   : vector;
    SampleScheme  : matrix;

{ Resource : Total # 1s in the matrix.
  ColumnSum : Maximum allowed # 1s among all columns.
  Depth : Used in the for loop.
  k : Maximum allowed # 1s in any row/column.
  Columns : This is a vector. Its I-th element gives the # 1s still allowed to
            be put in the I-th column.
  EmptyColumn : # columns which has no 1 so far.                    }

    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - Procedure Initialise - - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

procedure initialise;
var
    I,J : integer;
begin
     for I:= 1 to NoOfRows do
         for J := 1 to NoOfColumns do
             SampleScheme [I,J] := 0;
     NoOfSolutions := RequiredNumber;
     for J:= 1 to NoOfColumns do
         Columns [J] := k;
     ColumnSum := k * NoOfColumns;
     EmptyColumn := NoOfColumns;
     Depth := k - 1
end;

    (* 1 ------------------------------------------------------------1 *)
    (* 1 - - - - - - - - - Procedure Generate - - - - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

{This procedure generates the solution matrix in a systematic way.}

procedure generate(SampleMatrix:matrix; Columns:vector; ColumnSum,Resource,
                   NoOfRows,EmptyColumn : integer);
var
  J,J1,Remaining : integer;
  Possible : boolean;
{ Possible : True if it is possible to complete the matrix with the remaining
             1's , subject to the constraints.
 Remaining : When we fill up the row 1 , we must put 1's in those columns
             which has not yet obtained its minimum requirement, and  the
             rest of resource is called then "Remaining".                 }

        (* 2 ------------------------------------------------------ 2 *)
        (* 2 - - - - - -Procedure For_Loop - - - - - - - - - - - - 2 *)
        (* 2 ------------------------------------------------------ 2 *)

  { This procedure creates a nesting of for-loops, where the maximum level
  of nesting is Depth.If the maximum # allowed 1s is k, then in a particular
  row there can be i 1s, 2 <= i <= k in all possible ways. Hence for fixed
  i, we need i nested for loops and this procedure achieves that nesting
  for all i .

  procedure for_loop (LeftLimit,Depth:integer);
  var
     I,Temporary : integer;
  begin
       Temporary := Depth - k;
       generate(SampleMatrix,Columns,ColumnSum+Temporary,Resource+Temporary,
                NoOfRows-1,EmptyColumn);
       for I:= LeftLimit to
```

```
      If Depth > 0
      then begin
                Depth := Depth - 1;
                for I := LeftLimit to NoOfColumns do
                    if Columns[I] <> 0
                    then begin
                              SampleMatrix[NoOfRows,I] := 1;
                              Columns[I] := Columns[I] - 1;
                              if Columns[I] = (k - 1)
                              then
                                   EmptyColumn := EmptyColumn - 1;
                              for_loop (I+1,Depth);
                              SampleMatrix[NoOfRows,I] := 0;
                              Columns[I] := Columns[I] + 1;
                              if Columns[I] = k
                              then
                                   EmptyColumn := EmptyColumn + 1;
                         end
           end
end; (for_loop)

      (* 2 ------------------------------------------------------ 2 *)
      (* 2 - - - - - -Procedure First_Row_For_Loop - - - - - - - 2 *)
      (* 2 ------------------------------------------------------ 2 *)

{ This procedure is similar to For_Loop procedure with the difference that
it fills up the first row (last in our sequence) and hence sometimes
prints the matrix.}

procedure first_row_for_loop( LeftLimit,HowMany : integer );
{HowMany: Gives the remaining number of entries to be filled up in the first row }
var
   J : integer ;
begin
     if HowMany = 1
     then
         for J := LeftLimit to NoOfColumns do
             if (Columns[J] <> 0) and (SampleMatrix[1,J] = 0)
             then begin
                  SampleMatrix[1,J] := 1;
                  NoOfSolutions := NoOfSolutions - 1; {count down}
                  if NoOfSolutions < 0.1 {Basically we are testing = 0,but since it is a }
                  then                   {real number so scope of approximation is kept. }
                      Construct(SampleMatrix);
                  SampleMatrix[1,J] := 0
                  end
                  else
        else
            for J := LeftLimit to NoOfColumns do
                if (Columns[J] <> 0) and (SampleMatrix[1,J] = 0)
                then begin
                     SampleMatrix[1,J] := 1;
                     first_row_for_loop (J + 1 , HowMany - 1);
                     SampleMatrix[1,J] := 0;
                     end
   end; (first_row_for_loop)

    (* 1 - - - - - - - - - body of generate - - - - - - - - - - - - - 1 *)
begin (generate)
     Possible := false;
     if Resource >= NoOfRows  {Since each row/column must have at least a 1.}
     then if Resource >= EmptyColumn
          then if Resource <= (k * NoOfRows)
               then if Resource <= ColumnSum {Remember initially ColumnSum = k * NoOfColumns.}
                    then Possible := true;
     if Possible
     then
         if NoOfRows=1
         then begin
                   for J := 1 to NoOfColumns do
                       if Columns[J] = k
                       then SampleMatrix[1,J] := 1; {Since each column must have at least two 1's.}
                  Remaining := Resource - EmptyColumn;
```

```
            if Remaining = 0
            then begin (So, a solution is obtained.)
                      NoOfSolutions := NoOfSolutions - 1; (count down)
                      if NoOfSolutions < 0.1
                      then Construct(SampleMatrix)
                 end
            else (Put the remaining 1's in all possible ways.)
                first_row_for_loop (i,Remaining)
      end
else
    for J:= 1 to NoOfColumns do
                if Columns[J] <> 0
                then begin
                          SampleMatrix[NoOfRows,J] := 1;
                          Columns[J] := Columns[J] -1;
                          if Columns[J] = (k-1)
                          then
                               EmptyColumn := EmptyColumn - 1;
                          for_loop (J+1,Depth);
                          SampleMatrix[NoOfRows,J] := 0;
                          Columns[J] := Columns[J] +1;
                          if Columns[J] = k
                          then
                               EmptyColumn := EmptyColumn + 1;
                     end
end; (generate)

{ - - - - - - - - - - body of GenerateMatrix - - - - - - - - - - - - - - - - }
begin (main)
          Resource := NoOfRows * k;
          initialise;
          writeln(Outfile,'The solution matrix is');
          generate(SampleScheme,Columns,ColumnSum,Resource,NoOfRows,EmptyColumn);
end (of GenerateMatrix);


{-------------------------------------------------------------------------------}
{- - - - - - - - - - - - - - procedure ChooseSample - - - - - - - - - - - - - -}
{-------------------------------------------------------------------------------}

{This procedure chooses M * k sample units from N population units, in three
 ways depending on the value of the Remainder.}

procedure ChooseSample;
var
   I : integer;
   C : vector;
   RandomNumber,Total,RequiredNumber : real;
    (* 1 ------------------------------------------------------------ 1 *)
    (* 1 - - - - - - - - - - procedure OrdinaryScheme - - - - - - - - - 1 *)
    (* 1 ------------------------------------------------------------ 1 *)

{This procedure chooses M rows and M columns from the R X R matrix at random.
 Then it finds the # ways in which this M X M submatrix can be filled up by
 M * k 1s. Finally it finds any such matrix at random .}

procedure OrdinaryScheme;
var
   I : integer;
   C : vector;
begin
     writeln(Outfile);
     for I := 1 to (K - 1) do
           C[I] := 0;
     C[K] := M;
     count(M,K,C,Total);
     writeln(Outfile,'Total # solutions =',Total : 8:0);
     writeln('Which solution is required ?');
     write('(Input an integer between 1 and Total # solutions) ');
     readln(RequiredNumber);
```

```
      then begin
                Depth := Depth - 1;
                for I := LeftLimit to NoOfColumns do
                    if Columns[I] <> 0
                    then begin
                              SampleMatrix[NoOfRows,I] := 1;
                              Columns[I] := Columns[I] - 1;
                              if Columns[I] = (k - 1)
                              then
                                   EmptyColumn := EmptyColumn - 1;
                              for_loop (I+1,Depth);
                              SampleMatrix[NoOfRows,I] := 0;
                              Columns[I] := Columns[I] + 1;
                              if Columns[I] = k
                              then
                                   EmptyColumn := EmptyColumn + 1;
                         end
           end
end; (for_loop)

      (* 2 ------------------------------------------------------ 2 *)
      (* 2 - - - - - -Procedure First_Row_For_Loop - - - - - - - 2 *)
      (* 2 ------------------------------------------------------ 2 *)

( This procedure is similar to For_Loop procedure with the difference that
it fills up the first row (last in our sequence) and hence sometimes
prints the matrix.)

procedure first_row_for_loop( LeftLimit,HowMany : integer );
(HowMany: Gives the remaining number of entries to be filled up in the first row )
var
   J : integer ;
begin
     if HowMany = 1
     then
         for J := LeftLimit to NoOfColumns do
             if (Columns[J] <> 0) and (SampleMatrix[1,J] = 0)
             then begin
                  SampleMatrix[1,J] := 1;
                  NoOfSolutions := NoOfSolutions - 1; (count down)
                  if NoOfSolutions < 0.1 (Basically we are testing = 0,but since it is a )
                  then                    (real number so scope of approximation is kept. )
                      Construct(SampleMatrix);
                  SampleMatrix[1,J] := 0
                  end
             else
     else
         for J := LeftLimit to NoOfColumns do
             if (Columns[J] <> 0) and (SampleMatrix[1,J] = 0)
             then begin
                  SampleMatrix[1,J] := 1;
                  first_row_for_loop (J + 1 , HowMany - 1);
                  SampleMatrix[1,J] := 0;
                  end
   end; (first_row_for_loop)

    (* 1 - - - - - - - - - body of generate - - - - - - - - - - - - - - 1 *)
begin (generate)
     Possible := false;
     if Resource >= NoOfRows  (Since each row/column must have at least a 1.)
     then if Resource >= EmptyColumn
          then if Resource <= (k * NoOfRows)
               then if Resource <= ColumnSum (Remember initially ColumnSum = k * NoOfColumns.)
                    then Possible := true;
     if Possible
     then
         if NoOfRows=1
         then begin
                  for J := 1 to NoOfColumns do
                      if Columns[J] = k
                      then SampleMatrix[1,J] := 1; (Since each column must have at least two 1's.)
                  Possible := Resource = EmptyColumn;
```

```
' The above three lines can be replaced by the following line, to make the choice of solution automatic.');
  RequiredNumber := int (random * Total) + 1.0; }

     writeln(RequiredNumber : 8:0,'-th solution is selected.');
     Pattern := 1;
     GenerateMatrix (M,M,K,RequiredNumber)
end;

{ - - - - - - - - - - - - body of ChooseSample - - - - - - - - - - - - }
begin
     case Remainder of
     0: begin
        writeln(Outfile);
        writeln(Outfile,'Design D1 is selected.');
        OrdinaryScheme
        end;
     1: begin
        RandomNumber := random;
        writeln(Outfile);
        write(Outfile,'In case 1, RandomNumber = ',RandomNumber:6:4,'  & ');
        if RandomNumber < f
        then begin
             writeln(Outfile);
             writeln(Outfile,'Design D21 is selected.');
             OrdinaryScheme
             end
        else begin
             writeln(Outfile);
             writeln(Outfile,'Design D22 is selected.');
             for I := 1 to (K - 2) do
                 C[I] := 0;
             C[K - 1] := R;
             count(R,K - 1,C,Total);
             writeln(Outfile,'Total # solutions =',Total : 8:0);
             writeln('Which solution is required ?');
             write('(Input an integer between 1 and Total # solutions) ');
             readln(RequiredNumber);

{ The above three lines can be replaced by the following line, to make the choice of solution automatic.');
  RequiredNumber := int (random * Total) + 1.0; }

             writeln(RequiredNumber : 8:0,'-th solution is selected.');
             Pattern := 2;
             GenerateMatrix (R,R,K - 1,RequiredNumber)
             end
        end
     else begin
          RandomNumber := random;
          writeln(Outfile);
          write(Outfile,'In case else, RandomNumber = ',RandomNumber:6:4,'  & ');
          if RandomNumber < f
          then begin
               writeln(Outfile);
               writeln(Outfile,'Design D31 is selected.');
               OrdinaryScheme
               end
          else
              begin
              writeln(Outfile);
              writeln(Outfile,'Design D32 is selected.');
              for I := 1 to (K - 2) do
                  C[I] := 0;
              C[K - 1] := R;
              count(R,K - 1,C,Total);
              writeln(Outfile,'Total # solutions =',Total : 8:0);
              writeln('Which solution is required ?');
              write('(Input an integer between 1 and Total # solutions) ');
              readln(RequiredNumber);

{ The above three lines can be replaced by the following line, to make the choice of solution automatic.');
  RequiredNumber := int (random * Total) + 1.0; }
```

```
                    writeln('RequiredNumber = ',RequiredNumber);
                    Pattern := 3;
                    GenerateMatrix (R,R,K - 1,RequiredNumber)
                    end
                end
        end (case)
end; (of ChooseSample)

{------------------------ main program ------------------------------}
begin (of main)
        ScreenDesign;
        clrscr;
        write('Output file name =?(Type con for output on screen) ');
        readln(FileName);
        assign(Outfile,FileName);
        rewrite(Outfile);

        InputParameters;
        ChooseSample;

end (of main).
```

s selected.');