Pak J Commer Soc Sci
Pakistan Journal of Commerce and Social Sciences
2013, Vol. 7 (3), 564-568

An Algorithm to Check Balance Property of a Design

Rashid Ahmed (Corresponding Author)
Government Higher Secondary School Mitroo, Vehari, Pakistan
E-mail: rashid701@hotmail.com

M. Yousuf Aziz
Department of Statistics, The Islamia University of Bahawalpur, Pakistan
E-mail: myousufaziz@ymail.com

M.H. Tahir
Department of Statistics, The Islamia University of Bahawalpur, Pakistan
E-mail: mtahir.stat@gmail.com

Abstract

Balance incomplete block design and neighbor balanced deigns have their own importance in the field experiments such as agriculture, horticulture and forestry, etc. In the literature, there are several algorithms to construct the neighbor balanced designs but in this study, an algorithm is developed to check whether a given design is (i) a balanced incomplete block design, (ii) a neighbor-balanced, if so up to which order. This algorithm is coded in Visual C++.

Keywords: balanced incomplete designs; first order neighbor balanced designs; second order neighbor designs; incidence matrix.

1. Introduction

Neighbor design is a collection of circular blocks in which any two distinct treatments appear as neighbors equally often. Neighbor balanced designs ensure that treatment comparisons will be as little affected by competition/neighbor effects as possible. Experiments in agriculture, horticulture and forestry often show neighbor effects. Rees (1967) introduced neighbor designs in serology and constructed these designs for all odd v (number of treatments) when k (block size) = v. He also presented neighbor designs for odd v up to 41 with k $\leq$ 10. Some other references are discussed in detail by Ahmed et al. (2011).

Ahmed and Akhtar (2012) developed an algorithm to search possible first, second and higher order neighbor balanced designs for k = v-1. This algorithm is coded in Visual C++. In this article, an algorithm is developed to check whether a given design is (i) a balanced incomplete block design, (ii) a neighbor-balanced, if so up to which order. This algorithm is presented in Section 2 which is coded in visual C++ in Section 3.

2. An Algorithm to Check Neighbor Balance Property of a Design

This algorithm is to check whether the given design is a BIBD or not, and/or the given design is a neighbor-balanced design or not. If it is, then up to which order. The steps are:

1. Identify v (the number of treatments), b (the number of blocks) and k (the block size), and initialize $p = 1$.
2. Create **X**, the incidence matrix of the treatment effects of order (bk×v).
3. Compare the off-diagonal elements of concurrence matrix $\mathbf{NN^t} = \mathbf{X^tX}$. If all off-diagonal elements are same, the design is BIBD.
4. Create $\mathbf{X_1}$, incidence matrix for *pth*-order neighbor effects of order (bk×v).
5. Compare the off-diagonal elements of $\mathbf{X^tX_1}$. If all are same then the design is *pth*-order neighbor balanced goto step 6, otherwise stop.
6. Set $p = p + 1$ and go to step 4, otherwise stop.

3. Program in Visual C++ to Search Properties of a Design

```
// A Program to check the balance property of given design
//*****************************
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
int
c,d,v,m,k,i,j,a[80][105],n[80][105]={0},nn[80][105]={0};
int
x[750][80]={0},x1[750][80]={0},xx1[80][80]={0},l,t1,p,b,w=0
;
cout<<"enter v (number of treatments),k (block size), b
(number of blocks)."<<endl;
cin>>v>>k>>b;
cout<<"Label the v treatments as 0, 1, 2, ..., v-1"<<endl;
cout<<"Columns represent the blocks"<<endl;
cout<<"Enter the design values row-wise"<<endl;
for(j=0;j<b;j++)
for(i=0;i<k;i++)
{     cout<<"Enter d("<<j+1<<", "<<i+1<<")  "<<endl;
      cin>>a[i][j];
      l=a[i][j];  n[l][j]=1;
}
cout<<"Design for v = "<<v<<", k = "<<k<<", b = "<<b<<endl;
cout<<"**********************************"<<endl;
for(i=0;i<k;i++)
{     for(j=0;j<b;j++)
{     cout<<a[i][j]<<"  "; }
cout<<endl;
}
cout<<"NN' matrix"<<endl;
cout<<"**********"<<endl;
```

```
for(i=0;i<v;i++)
for(j=0;j<v;j++)
for(m=0;m<b;m++)
nn[i][j]+=n[i][m]*n[j][m];
for(i=0;i<v;i++)
{
for(j=0;j<v;j++)
cout<<nn[i][j]<<"  ";
cout<<"\n";
}
for(d=0;d<b;d++)
for(c=0;c<k;c++)
for(j=0;j<v;j++)
{     if(a[c][d]==j) x[c+k*d][j]=1;
else
x[c+k*d][j]=0;
}
for(i=0;i<v;i++)
{
for(j=0;j<v;j++)
if(i!=j)
{if(nn[i][j]!=nn[v-1][v-2]) goto rs2;}
}
cout<<endl;
cout<<"Design is BIBD with Lambda = "<<nn[v-1][v-2]<<endl;
goto ra1;
rs2:
cout<<"Design is not BIBD"<<endl;
ra1: p=1; cout<<endl;
ra:
for(c=0;c<b;c++)
for(j=0;j<k;j++)
{
if(j+p>=k)
{
t1=a[j-k+p][c];
x1[j+k*c][t1]=1;
}
else
{
t1=a[j+p][c];
x1[j+k*c][t1]+=1;
}
if(j+k-p>=k)
{
t1=a[j-p][c];
x1[j+k*c][t1]+=1;
}
else
{
```

```
t1=a[j+k-p][c];
x1[j+k*c][t1]+=1;
}}
cout<<"X'X<<p<<"matrix for "<<p<<"-th order neighbor"<<
endl;
cout<<"***************"<<endl;
for(i=0;i<v;i++)
for(j=0;j<v;j++)
for(m=0;m<k*b;m++)
xx1[i][j]+=x[m][i]*x1[m][j];
for(i=0;i<v;i++)
{
for(j=0;j<v;j++)
cout<<xx1[i][j]<<"  ";
cout<<"\n";
}
for(i=0;i<v;i++)
  {    for(j=0;j<v;j++)
        if(i!=j)
      {     if(xx1[i][j]!=xx1[v-1][v-2]) goto rs; }
}
cout<<"Design is "<<p<<" order neighbor balance with
Lambda("<<p<<") = ";
cout <<xx1[v-1][v-2]<<endl; cout<<endl;
cout<<"Enter 10 if further neighbor balance is to be
checked"<<endl;
cout<<"otherwise any integer except 10"<<endl; cout<<endl;
cin>>w;
if(w==10)
{     for(i=0;i<v;i++)
for(j=0;j<v;j++) xx1[i][j]=0;
for(i=0;i<k*b;i++)
for(j=0;j<v;j++) x1[i][j]=0;
p=p+1;if(p<=int(k/2)) goto ra;
else goto rs1;
}goto rs;
rs1:
cout<<"Design is all order neighbor balanced"<<endl;rs:
getch();
}
```

REFRENCES

Rees, D. H. (1967). Some designs of use in serology, *Biometrics*, 23, 779–791.

Ahmed, R., Akhtar, M. and Yasmin, F. (2011). Brief review of one dimensional neighbor balanced designs since 1967. *Pakistan Journal of Commerce and Social Sciences*, 5(1), 100-116.

Ahmed, R. and Akhtar, M. (2012). Computer-generated neighbor designs. *Communication in Statistics - Simulation and Computation*, 41, 1834-1839.