%MktBIBD Macro

The `%MktBIBD` autocall macro finds balanced incomplete block designs (BIBDs). A BIBD is a list of treatments that appear together in blocks. Each block contains a subset of the treatments. BIBDs can be used in marketing research to construct partial-profile designs (Chrzan and Elrod 1995). The entries in the BIBD say which attributes are to be shown in each set. For example, a BIBD could be used for the situation where there are t attributes or messages (treatments) and k are shown at a time. A total of b sets of attributes (blocks) are shown. For examples of using this macro in the partial-profile context, see page 1145. See the following pages for examples of using this macro in the discrete choice chapter: 642, 645, 650, 654, 656, 659 and 662. Additional examples appear throughout this chapter.

BIBDs are also used in marketing research to construct MaxDiff (best-worst) designs (Louviere 1991, Finn and Louviere 1992). In a MaxDiff study, subjects are shown sets (blocks) of messages or product attributes (treatments) and are asked to choose the best (or most important) from each set as well as the worst (or least important). For examples of using this macro in the MaxDiff context, see page 1105.

The documentation for the `%MktPPro` and `%MktMDiff` macros discuss the `%MktBIBD` macro in the context of sets and attributes, whereas most of the `%MktBIBD` macro documentation uses the more traditional statistical vocabulary of blocks and treatments. Furthermore, the output of the `%MktBIBD` macro can correspond to the marketing research vocabulary (when the `nattrs=` option is specified) or the statistical vocabulary (when the `t=` option is specified). The `t=` and `nattrs=` options are otherwise aliases for each other and specify the number of treatments (or attributes).

The following design is an example of a BIBD:

Balanced Incomplete Block Design

x1	x2	x3
3	5	2
1	5	2
3	2	4
4	3	1
5	3	1
1	2	3
4	1	5
5	4	3
2	1	4
2	4	5

This BIBD has $b = 10$ blocks of treatments. Each of the 10 rows is a block. There are $t = 5$ treatments, since the entries in the design are the integers 1 through 5. Each block consists of $k = 3$ treatments. Note that the number of entries in the design, $b \times k = 10 \times 3 = 6 \times t = 30$. Each of the $t = 5$ treatments occurs exactly 6 times in the design, and each treatment occurs with every other treatment 3 times. This can be seen in the following treatment by treatment frequencies:

Treatment by Treatment Frequencies

	1	2	3	4	5
1	6	3	3	3	3
2		6	3	3	3
3			6	3	3
4				6	3
5					6

When the %MktBIBD macro makes a BIBD, it tries to optimize the treatment by position frequencies. In other words, it tries to ensure that each treatment occurs in each of the k positions equally often, or at least close to equally often. The following are the treatment by position frequencies for this design, which in this case are perfect:

Treatment by Position Frequencies

	1	2	3
1	2	2	2
2	2	2	2
3	2	2	2
4	2	2	2
5	2	2	2

The following %MktBIBD macro step generates this BIBD:

```
%mktbibd(b=10, t=5, k=3, seed=104)
```

In addition to the three tables above, the %MktBIBD macro also displays the following summary information:

Block Design Efficiency Criterion	100.0000
Number of Treatments, t	5
Block Size, k	3
Number of Blocks, b	10
Treatment Frequency	6
Pairwise Frequency	3
Total Sample Size	30
Positional Frequencies Optimized?	Yes

The fact the that efficiency is 100 shows that %MktBIBD found a BIBD. In many cases, it will find a design that is close to a BIBD but each of the pairwise frequencies is not constant. For many marketing

research problems, this is good enough.

The BIBD is available in the `out=BIBD` data set. The following step displays the design:

```
proc print data=bibd noobs; run;
```

The results are as follows:

x1	x2	x3
3	5	2
1	5	2
3	2	4
4	3	1
5	3	1
1	2	3
4	1	5
5	4	3
2	1	4
2	4	5

Every BIBD has a binary representation, or *incidence matrix*, with b rows and t columns and k ones that indicate which treatments appear in each block. The following step displays the `outi=incidence` matrix:

```
proc print data=incidence noobs; run;
```

The results are as follows:

b1	b2	b3	b4	b5
0	1	1	0	1
1	1	0	0	1
0	1	1	1	0
1	0	1	1	0
1	0	1	0	1
1	1	1	0	0
1	0	0	1	1
0	0	1	1	1
1	1	0	1	0
0	1	0	1	1

This shows that the first block consists of treatments 2, 3, and 5, just as the `out=BIBD` design does, but in a different format.

You can also view the BIBD arrayed as a block by treatment factorial design. The following step displays the first 9 (out of 30) observations:

```
proc print data=factorial(obs=9) noobs; run;
```

The results are as follows:

Block	Treatment
1	3
1	5
1	2
2	1
2	5
2	2
3	3
3	2
3	4

The following %MktBIBD macro step generates a BIBD with nondirectional row-neighbor balance:

```
%mktbibd(b=14, t=7, k=4, options=neighbor, seed=104)
```

The results are as follows:

Block Design Efficiency Criterion	100.0000
Number of Treatments, t	7
Block Size, k	4
Number of Blocks, b	14
Treatment Frequency	8
Pairwise Frequency	4
Total Sample Size	56
Positional Frequencies Optimized?	Yes
Row-Neighbor Frequencies Optimized?	Yes

Treatment by Treatment Frequencies

	1	2	3	4	5	6	7
1	8	4	4	4	4	4	4
2		8	4	4	4	4	4
3			8	4	4	4	4
4				8	4	4	4
5					8	4	4
6						8	4
7							8

Treatment by Position Frequencies

	1	2	3	4
1	2	2	2	2
2	2	2	2	2
3	2	2	2	2
4	2	2	2	2
5	2	2	2	2
6	2	2	2	2
7	2	2	2	2

Row-Neighbor Frequencies

	1	2	3	4	5	6	7
1		2	2	2	2	2	2
2			2	2	2	2	2
3				2	2	2	2
4					2	2	2
5						2	2
6							2
7							

Balanced Incomplete Block Design

x1	x2	x3	x4
3	4	2	5
5	6	1	4
1	4	5	3
7	1	6	4
4	5	7	2
5	3	7	6
3	7	5	1
1	2	4	7
6	2	1	5
4	7	6	3
2	3	4	6
2	6	3	1
6	5	2	7
7	1	3	2

The resulting design is a BIBD with each of the 7 treatments appearing in each of the 4 positions within a block exactly twice. Furthermore, the row-neighbor frequencies show that each of the $7\times(7-1)/2 = 21$ pairs of treatments occur exactly twice. The pairs in this design are 3 with 4, 4 with 2, 2 with 5, 5 with 6, and so on. The order of the treatments in each pair is ignored. Hence, in this design, with the nondirectional row-neighbor balance, the 2 followed by 5 in the first row of the design is treated the same as the 5 followed by 2 in the second to last row of the design. This design is usually easily found

in a few seconds. The last line of the first table ("Row-Neighbor Frequencies Optimized? Yes") along with the constant row-neighbor frequencies, shows that perfect row-neighbor balance was achieved.

The following finds a BIBD with row-neighbor balance where the order of the treatments *does* matter using `options=serial`:

```
%mktbibd(b=14, t=7, k=4, options=serial, seed=361699)
```

The results are as follows:

Block Design Efficiency Criterion	100.0000
Number of Treatments, t	7
Block Size, k	4
Number of Blocks, b	14
Treatment Frequency	8
Pairwise Frequency	4
Total Sample Size	56
Positional Frequencies Optimized?	Yes
Row-Neighbor Frequencies Optimized?	Yes

Treatment by Treatment Frequencies

	1	2	3	4	5	6	7
1	8	4	4	4	4	4	4
2		8	4	4	4	4	4
3			8	4	4	4	4
4				8	4	4	4
5					8	4	4
6						8	4
7							8

Treatment by Position Frequencies

	1	2	3	4
1	2	2	2	2
2	2	2	2	2
3	2	2	2	2
4	2	2	2	2
5	2	2	2	2
6	2	2	2	2
7	2	2	2	2

Row-Neighbor Frequencies

	1	2	3	4	5	6	7
1		1	1	1	1	1	1
2	1		1	1	1	1	1
3	1	1		1	1	1	1
4	1	1	1		1	1	1
5	1	1	1	1		1	1
6	1	1	1	1	1		1
7	1	1	1	1	1	1	

Balanced Incomplete Block Design

x1	x2	x3	x4
1	4	2	5
3	1	5	7
4	5	1	7
2	3	7	1
6	7	5	3
7	6	4	1
7	2	6	5
1	3	4	6
3	5	2	4
4	7	3	2
5	4	3	6
5	6	1	2
6	2	7	4
2	1	6	3

In this set of output, with `options=serial`, the row-neighbor frequencies appear both above and below the diagonal. In contrast, in the previous set of output, with `options=neighbor`, the row-neighbor frequencies only appeared above the diagonal. You can see that each pair occurs exactly once in this design. In this design, with the serial (directional) row-neighbor balance, the 2 followed by 5 in the first row of the design is *not* treated the same as the 5 followed by 2 in the ninth row of the design.

An `options=serial` design is typically much harder to find than an `options=neighbor` design. In this case, a random number seed that is known to produce an optimal design reasonably quickly was chosen.* Usually, you will have to run the macro more than once or change some options such as the time value in the `positer=` option and iterate for up to a few hours to find an equivalent design.

The following requests this same design with a different seed:

```
%mktbibd(b=14, t=7, k=4, options=serial, seed=104)
```

*By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, results might not be exactly reproducible everywhere.

A portion of the output is as follows:

```
Positional Frequencies Optimized?            No
Row-Neighbor Frequencies Optimized?          No

       Treatment by Position Frequencies

              1  2  3  4

          1   2  2  2  2
          2   2  3  2  1
          3   2  2  2  2
          4   2  2  2  2
          5   2  2  2  2
          6   2  2  2  2
          7   2  1  2  3

      Row-Neighbor Frequencies

          1  2  3  4  5  6  7

       1     1  1  1  1  1  1
       2  1     1  1  1  2  1
       3  1  1     1  1  1  1
       4  1  1  1     1  1  1
       5  1  1  1  1     1  1
       6  1  1  1  1  1     1
       7  1  1  1  1  1  0
```

With most seeds and the default amount of iteration you will get results like these with nearly optimal position and row neighbor frequencies.

The following requests a design where it is not possible to have constant frequencies in the row-neighbor frequencies matrix:

```
%mktbibd(b=7, t=7, k=4, options=serial, seed=104)
```

A portion of the output is as follows:

```
Positional Frequencies Optimized?            Yes
Row-Neighbor Frequencies Optimized?          Yes
```

Row-Neighbor Frequencies

	1	2	3	4	5	6	7
1		0	1	1	1	0	0
2	0		1	0	0	1	1
3	1	0		0	0	1	1
4	0	1	0		1	1	0
5	1	1	0	0		0	1
6	0	0	1	1	1		0
7	1	1	0	1	0	0	

The %MktBIBD macro reports that the row-neighbor frequencies are optimized since a mix of zeros and ones with no twos or larger values is optimal for this specification. However, the nonconstant frequencies show that row-neighbor balance is not possible for this BIBD.

BIBD Parameters

The parameters of a BIBD are:

- b specifies the number of blocks. In a partial-profile design, this is the number of profiles. In a MaxDiff design, this is the number of sets.
- t specifies the number of treatments. In a partial-profile or MaxDiff design, this is the total number of attributes or messages.
- k specifies the block size, which is the number of treatments in each block. In a partial-profile or MaxDiff design, this is the number of attributes or messages shown at one time.

When $r = b \times k/t$ and $l = r \times (k-1)/(t-1)$ are integers, and $k = t$ and $b \geq t$, then a complete block design might be possible. This is a necessary but not sufficient condition for the existence of a complete block design. When $r = b \times k/t$ and $l = r \times (k-1)/(t-1)$ are integers, and $k < t$ and $b \geq t$, then a balanced incomplete block design might be possible. This is a necessary but not sufficient condition for the existence of a BIBD. You can use the macro %MktBSize to find parameters in which BIBDs might exist. The %MktBIBD macro uses PROC OPTEX and a computerized search to find BIBDs. It does not have a library of BIBDs or use combinatorial constructions. Hence, it will not always find a BIBD even when one is known to exist. However, it usually works quite well in finding small BIBDs or something close for larger problems. The following step displays some of the smaller specifications for which a BIBD might exist:

```
%mktbsize(t=1 to 20, k=2 to 0.5 * t, b=t to 100)
```

The results are as follows:

t Number of Treatments	k Block Size	b Number of Blocks	r Treatment Frequency	Lambda Pairwise Frequencies	n Total Sample Size
4	2	6	3	1	12
5	2	10	4	1	20
6	2	15	5	1	30
6	3	10	5	2	30
7	2	21	6	1	42
7	3	7	3	1	21
8	2	28	7	1	56
8	3	56	21	6	168
8	4	14	7	3	56
9	2	36	8	1	72
9	3	12	4	1	36
9	4	18	8	3	72
10	2	45	9	1	90
10	3	30	9	2	90
10	4	15	6	2	60
10	5	18	9	4	90
11	2	55	10	1	110
11	3	55	15	3	165
11	4	55	20	6	220
11	5	11	5	2	55
12	2	66	11	1	132
12	3	44	11	2	132
12	4	33	11	3	132
12	6	22	11	5	132
13	2	78	12	1	156
13	3	26	6	1	78
13	4	13	4	1	52
13	5	39	15	5	195
13	6	26	12	5	156
14	2	91	13	1	182
14	4	91	26	6	364
14	6	91	39	15	546
14	7	26	13	6	182
15	3	35	7	1	105
15	5	21	7	2	105
15	6	35	14	5	210
15	7	15	7	3	105

16	3	80	15	2	240
16	4	20	5	1	80
16	5	48	15	4	240
16	6	16	6	2	96
16	7	80	35	14	560
16	8	30	15	7	240
17	4	68	16	3	272
17	5	68	20	5	340
17	8	34	16	7	272
18	6	51	17	5	306
18	9	34	17	8	306
19	3	57	9	1	171
19	4	57	12	2	228
19	6	57	18	5	342
19	7	57	21	7	399
19	9	19	9	4	171
20	4	95	19	3	380
20	5	76	19	4	380
20	8	95	38	14	760
20	10	38	19	9	380

%MktBIBD Macro Options

The following options can be used with the `%MktBIBD` macro:

Option	Description
`help`	(positional) "help" or "?" displays syntax summary
`b=`*b*	number of blocks (alias for `nsets=`)
`group=`*n*	number of groups
`k=`*k*	block size (alias for `setsize=`)
`nattrs=`*t*	number of attributes (alias for `t=`)
`nsets=`*b*	number of sets (alias for `b=`)
`optiter=`*n1 < n2 < n3 >>*	number of PROC OPTEX iterations
`options=position`	optimizes position frequencies
`options=neighbor`	nondirectional row-neighbor balance and position
`options=serial`	directional row-neighbor balance and position
`out=`*SAS-data-set*	output data set BIBD
`outf=`*SAS-data-set*	output data set factorial design
`outi=`*SAS-data-set*	output data set incidence matrix
`outs=`*SAS-data-set*	output data set sorted design
`positer=`*n1 < n2 < n3 >>*	number of iterations position frequencies
`seed=`*n*	random number seed
`setsize=`*k*	set size (alias for `k=`)
`t=`*t*	number of treatments (alias for `nattrs=`)
`weights=`*n n*	position frequency badness weights

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktbibd(help)
%mktbibd(?)
```

b= *b*
nsets= *b*
specifies the number of blocks. In a partial-profile design, this is the number of profiles. In a MaxDiff design, this is the number of sets. The `nsets=` and `b=` options are aliases. This option (in one of its two forms) must be specified.

group= *n*
specifies the number of groups into which the design is to be divided. This could be useful with MaxDiff and partial profiles. By default, the design is not divided into groups.

k= *k*
setsize= *k*
specifies the block size, which is the number of treatments in each block. In a partial-profile or MaxDiff design, this is the number of attributes or messages shown at one time in each set. The `setsize=` and `k=` options are aliases. This option (in one of its two forms) must be specified.

optiter= *n1 < n2 < n3 >>*
specifies the number of PROC OPTEX iterations. If one number is specified, it is just the number of iterations. With two numbers, *n1* and *n2*, *n1* iterations are performed and then efficiency is checked. If the block design efficiency criterion is 100, the iterations stop. Otherwise, this process is repeated up to *n2* times for a maximum of $n1 \times n2$ iterations. The default for *n2*, when it is missing, is 1000 when the parameters conform to the necessary conditions for a BIBD and 5 otherwise. The third value is the maximum amount of time in minutes to spend in PROC OPTEX. The default for the time value, when it is missing, is 5 minutes when the parameters conform to the necessary conditions for a BIBD and 0.5 minutes otherwise. Hence, when the parameters conform to the necessary conditions for a BIBD, the default is `optiter=500 1000 5`, otherwise, the default is `optiter=500 5 0.5`. For larger problems, you might want to specify values smaller than the defaults.

options= *options-list*
specifies binary options. By default, `options=position`.

`position`
optimizes position frequencies. The goal is for each treatment to appear equally often in each position.

`neighbor`
optimizes nondirectional row-neighbor balance and also position. The goal is for pairs of treatments, constructed from each of the first $k-1$ treatments in each block along with the treatment that follows it, to occur equally often in the design. The order of the treatments within each pair does not matter in evaluating row-neighbor balance. That is, treatment 1 appearing before treatment 2 is counted as the same as 2 appearing before 1.

`serial`
optimizes directional row-neighbor balance and also position. The goal is for pairs of treatments, constructed from each of the first $k-1$ treatments in each block along with the treatment that follows it, to occur equally often in the design. In contrast to `options=neighbor`, treatment 2 appearing before treatment 1 is treated as different from 1 appearing before 2.

out= *SAS-data-set*
specifies the output data set name for the $b \times k$ BIBD (or more generally, the incomplete block design). The default is `out=BIBD`.

outf= *SAS-data-set*
specifies the output data set name for the $bk \times 2$ factorial design matrix. The default is `outf=Factorial`.

outi= *SAS-data-set*
specifies the output data set name for the $b \times t$ incidence matrix. The default is `outi=Incidence`.

outs= *SAS-data-set*
specifies the output data set name for the $b \times k$ sorted design. The default is `outs=Sorted`.

positer= *n1 < n2 < n3 >>*
specifies the number of iterations for the algorithm that attempts to optimize the treatment by position frequencies. The default is `positer=200 5000 2`. The first value specifies the number of times to try to refine the design. The second value specifies the number of times to start from scratch with a different random start. Larger values increase the chances of finding better treatment by position frequencies at a cost of slower run times. The third value is the maximum amount of time in minutes to spend optimizing the positional frequencies. Specify `positer=` or `positer=0` to just request a design without optimizing position.

seed= *n*
specifies the random number seed. By default, `seed=0`, and clock time is used to make the random number seed. By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system and for a particular version of the macro. However, due to machine and macro differences, some results might not be exactly reproducible everywhere, although you would expect the efficiency differences to be slight.

t= *t*
nattrs= *t*
specifies the number of treatments. In a partial-profile or MaxDiff design, this is the total number of attributes or messages. The `nattrs=` and `t=` options are aliases. This option (in one of its two forms) must be specified. When the `nattrs=` option is specified, the output will use the word "Attribute" rather than "Treatment" and "Set" rather than "Block".

weights= *n n*
specifies weights for position balance and row-neighbor balance. Specify two nonnegative numeric values. The total badness is a weighted sum of the position badness and the row-neighbor badness. The default is `weights=1 2`, so by default, row-neighbor balance is given the most weight. You can specify a weight of zero, for example, for the position balance (the first value) to just optimize row-neighbor balance.

Evaluating an Existing Block Design

The `%MktBIBD` macro does not have an option to evaluate existing block designs. However, you can run the following ad hoc macro to if you want to see the blocking design efficiency criterion, the treatment by position frequencies, and the treatment by treatment frequencies:

```
%macro mktbeval(design=_last_,      /* block design to evaluate         */
                attrortr=Attribute);/* string to print in the output    */
                                    /* specify Treatment or use default */

proc iml;
   use &design; read all into x;
   t = max(x);    k = ncol(x);    blocks = nrow(x);
   call symput('k', char(k));
   call symput('b', char(blocks));
   f = j(t, t, .);    p = j(t, k, 0);
   do i = 1 to blocks;
      do j = 1 to k;
         p[x[i,j],j] = p[x[i,j],j] + 1;
         do q = j to k;
            a = min(x[i,j],x[i,q]);
            b = max(x[i,j],x[i,q]);
            f[a,b] = sum(f[a,b], 1);
            end;
         end;
      end;
```

```
   options missing=' ';
   w = ceil(log10(t + 1)); t = right(char(1, w, 0) : char(t, w, 0));
   w = ceil(log10(k + 1)); q = right(char(1, w, 0) : char(k, w, 0));
   if max(f) < 100 then print "&attrortr by &attrortr Frequencies",,
      f[format=2. label='' rowname=t colname=t];
   else print "&attrortr by &attrortr Frequencies",,
      f[label='' rowname=t colname=t];
   if max(p) < 100 then print "&attrortr by Position Frequencies",,
      p[format=2. label='' rowname=t colname=q];
   else print "&attrortr by Position Frequencies",,
      p[label='' rowname=t colname=q];
   options missing='.';
   x = (1:blocks)` @ j(k, 1, 1) || shape(x, blocks * k);
   create __tmpbefac from x; append from x;
   quit;

proc optex;
   class col2;
   model col2;
   generate initdesign=__tmpbefac method=sequential;
   blocks structure=(&b)&k init=chain iter=0;
   ods select BlockDesignEfficiencies;
   run;

proc datasets nolist; delete __tmpbefac; run; quit;
%mend;
```

The macro has two options. Specify the name of the block design in the `design=` option. By default, the treatments are labeled as "Attributes", however, you can specify `attrortr=Treatment` if you would like them labeled as treatments.

The following steps create a BIBD and evaluate it:

```
%mktbibd(b=10, t=6, k=3, seed=104)

%mktbeval;
```

The results of the evaluation step are as follows:

Attribute by Attribute Frequencies

	1	2	3	4	5	6
1	5	2	2	2	2	2
2		5	2	2	2	2
3			5	2	2	2
4				5	2	2
5					5	2
6						5

Attribute by Position Frequencies

	1	2	3
1	2	2	1
2	2	1	2
3	2	2	1
4	1	2	2
5	1	2	2
6	2	1	2

The OPTEX Procedure

Design Number	Treatment D-Efficiency	Treatment A-Efficiency	Block Design D-Efficiency
1	80.0000	80.0000	100.0000

These results match the results from the `%MktBIBD` macro (not shown). The efficiency result of interest is the block design *D*-efficiency, with is 100 for BIBDs.

%MktBIBD Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.