DiGGer 1.0.0 Quick Guide

Neil Coombes

January 20, 2016

1 Introduction

`DiGGer` is a flexible tool for finding experimental designs that are efficient for specified blocking and correlation patterns. The package `DiGGer` (`http://www.austatgen.org/files/software/downloads`) is an add-on for the statistical computing language and environment R (R Development Core Team, 2009).

This major revision of DiGGer (Coombes, 2002) addresses several problems with the previous version

- default settings in the `DiGGer()` function, appropriate for its initial target of field trials with standard correlation, required substantial modifications when used with other design types
- the random number generator (RNG) was not programmed into the R version so that recovery of generated designs and illustration of the design process were not possible.

The package uses `lowerUpper` coding for variable and function names and `Upper` for constructor functions. The constructor functions in this version are:

- `DiGGer` which creates a DiGGer object with an initial design, initial swap matrix and treatment information. There is a slot set up for the search specifications and results.
- `Correlation` defines the plot correlation which may be identity, autoregressive or moving average.

- `Block` defines a rectangular block factor with row, column and variance ratio properties.
- `Objective` defines a search objective used in the optimisation process combining correlation and block objects.
- `Phase` defines a search phase for with swap, correlation, type of optimisation measure and the objectives to be optimised.

Wrapper functions use the constructor functions to perform searches for different design types. The wrapper functions in this version are:

- `corDiGGer` which can perform searches for spatially adjusted incomplete block designs in the manner of the `DiGGer` function in version 0.2-3.
- `ibDiGGer` for searches for incomplete block designs.
- `rcDiGGer` for row-column design searches.
- `prDiGGer` for partially replicated design searches.
- `facDiGGer` for searches for factorial and split plot designs.
- `r2dDiGGer` for searches with strict replication in two directions where replicates in the second set are not rectangular.

DiGGer was developed for field designs in rectangular arrays of plots so all design searches use specifications of rectangular layouts and block structures. For some of the design types listed these rectangular specifications may be nominal and may not correspond to actual physical layouts.

2 Examples

2.1 Incomplete Block Designs, `ibDiGGer`

Blocks to be optimised are specified by `rowsInBlock` and `columnsInBlock`. If `rowsInReplicate` and `columnsInReplicate` are supplied the design will be resolvable. Here is an example of a balanced incomplete block design with seeds that require a continuation of the search to find the optimum. In this case the `rngSeeds` are supplied. The optimal design for random blocks with variance ratio of 1.0 is 0.2857143 so a slightly larger target A value of 0.2857145 may be specified to cause the search to stop once the A value falls below this value. The maximum interchanges to test is increased from the 100000 default to 1000000.

```
ib19b <- ibDiGGer(numberOfTreatments = 19,
                  rowsInDesign = 57, columnsInDesign = 3,
                  rowsInBlock = 1, columnsInBlock = 3,
                  maxInterchanges = 1000000,
                  targetAValue = 0.2857145,
                  rngSeeds = c(9342, 18602),
                  runSearch = FALSE)
ib19b
ib19b <- run(ib19b)
```

```
getConcurrence(ib19b)

## B RANDOM BLOCK: 1 Row by 3 Columns, VarianceRatio 1.000000
##
## Treatment-Block
##
##   0   1   2
##   2 167   2
```

After this call to `ibDiGGer` there are 2 pairs of treatments that do not occur together in blocks and 2 pairs of treatments that occur together twice in blocks. There is an option in the `run` function to continue the search with the current best design. The search history records are cleared before continuing but the current settings of the RNG, `rngState`, are retained for repeatability of the search.

```
ib19b <- run(ib19b, continue = TRUE)
```

```
getConcurrence(ib19b)

## B RANDOM BLOCK: 1 Row by 3 Columns, VarianceRatio 1.000000
```

```
##
## Treatment-Block
##
##   1
## 171

ib19b$ddphase[[1]]$lastImprovement

## [1] 610171
```

After the continuation all 171 pairs of treatments occur together once in blocks. A total of 1610171 interchanges were used.

A balanced lattice design has resolvable blocks. Again the known optimum can be used as a stopping rule.

```
ib25s <- ibDiGGer(numberOfTreatments = 25,
                  rowsInDesign = 30, columnsInDesign = 5,
                  rowsInRep = 5, columnsInRep = 5,
                  rowsInBlock = 1, columnsInBlock = 5,
                  maxInterchanges = 10000000,
                  targetAValue = 0.387097,
                  rngSeeds = c(11301, 29798))

##      Phase,    Search%,    A-measure
## [1] 1.0000000 0.0000000 0.3966814
## [1] 1.0000000 5.0000000 0.3870968
##  [1] 0.3870968 0.3870968 0.0000000 0.0000000
##  [5] 0.0000000 0.0000000 0.0000000 0.0000000
##  [9] 0.0000000 0.0000000

getConcurrence(ib25s)

## B RANDOM BLOCK: 1 Row by 5 Columns, VarianceRatio 1.000000
##
## Treatment-Block
##
##   1
## 300
```

```
ib25s$ddphase[[1]]$lastImprovement

## [1] 520760
```

2.2 Row-Column Designs, `rcDiGGer`

Row-column searches can be undertaken in two phases or in a single phase. The rows and columns may extend across the full design or they may be nested, default `nested = TRUE`. The default search action is for two phase searches, `twoPhase = TRUE`, with the blocks with fewer plots optimised first. The order that the blocks are optimised can be controlled, `twoPhase = "rowThenCol"` or `twoPhase = "colThenRow"`.

These approaches are illustrated in a trial for 12 treatments in a $[\ 3 \times 12\ ]$ design with replicates $[\ 3 \times 4\ ]$. The default settings result in $[\ 3 \times 1\ ]$ column blocks being optimised before nested $[\ 1 \times 4\ ]$ row blocks. To help in efficiency calculations, `fixedBlocks = TRUE`.

```
test1 <- rcDiGGer(numberOfTreatments = 12,
                  rowsInDesign = 3, columnsInDesign = 12,
                  rowsInReplicate = 3, columnsInReplicate = 4,
                  maxInterchanges = 500000, fixedBlocks = TRUE,
                  rngSeeds = c(111, 222))
```

```
getConcurrence(test1, 2,1,1)

## B FIXED BLOCK: 1 Row by 4 Columns
##
## Treatment-Block
##
##  0  1  2
## 24 30 12

getConcurrence(test1, 2,1,2)

## B FIXED BLOCK: 3 Rows by 1 Column
##
```

```
## Treatment-Block
##
##  0  1
## 30 36

2/3/test1$ddphase[[2]]$aMeasures[1]

## [1] 0.5075503
```

The design efficiency is calculated from the fixed block A measure as $2/r/A$ where r is the number of replicates and A is the `DiGGer` fixed effect `aMeasure`. For the `test1` design it is 0.508. When row blocks are optimised before column blocks the the overall efficiency is reduced with row block concurrences better and column block concurrences worse.

```
test2 <- rcDiGGer(numberOfTreatments = 12,
                  rowsInDesign = 3, columnsInDesign = 12,
                  rowsInReplicate = 3, columnsInReplicate = 4,
                  fixedBlocks = TRUE, twoPhase = "rowThenCol",
                  rngSeeds = c(111, 222))
```

```
getConcurrence(test2, 2,1,1)

## B FIXED BLOCK: 1 Row by 4 Columns
##
## Treatment-Block
##
##  0  1  2
## 21 36  9

getConcurrence(test2, 2,1,2)

## B FIXED BLOCK: 3 Rows by 1 Column
##
## Treatment-Block
##
##  0  1  2
## 32 32  2
```

```
2/3/test2$ddphase[[2]]$aMeasures[1]

## [1] 0.5021073
```

By searching with `twoPhase = FALSE` the row concurrences are worse than the `"rowThenCol"` option. However continuing the search quickly finds an equivalent to the `test1` design.

```
test3 <- rcDiGGer(numberOfTreatments = 12,
                  rowsInDesign = 3, columnsInDesign = 12,
                  rowsInReplicate = 3, columnsInReplicate = 4,
                  fixedBlocks = TRUE,
                  twoPhase = "FALSE", rngSeeds = c(111, 222))
```

```
getConcurrence(test3, 1,1,1)

## B FIXED BLOCK: 1 Row by 4 Columns
##
## Treatment-Block
##
##  0  1  2
## 24 30 12

getConcurrence(test3, 1,1,2)

## B FIXED BLOCK: 3 Rows by 1 Column
##
## Treatment-Block
##
##  0  1  2
## 32 32  2

2/3/test3$ddphase[[1]]$aMeasures[1]

## [1] 0.5072034
```

```
test3 <- run(test3, continue = TRUE)
```

```
getConcurrence(test3, 1,1,1)

## B FIXED BLOCK: 1 Row by 4 Columns
##
## Treatment-Block
##
##  0  1  2
## 24 30 12

getConcurrence(test3, 1,1,2)

## B FIXED BLOCK: 3 Rows by 1 Column
##
## Treatment-Block
##
##  0  1
## 30 36

2/3/test3$ddphase[[1]]$aMeasures[1]

## [1] 0.5075503
```

An example of a non-resolvable row-column design for 80 entries in a [12×20] design is given in Venables and Eccleston (1993) with an efficiency bound of 0.8645

```
rc80 <- rcDiGGer(numberOfTreatments = 80,
                 rowsInDesign = 12, columnsInDesign = 20,
                 nested = FALSE, fixedBlock = TRUE,
                 rngSeeds = c(16867, 12675),
                 maxInterchanges = 500000, twoPhase = FALSE)
```

```
2/3/rc80$ddphase[[1]]$aMeasures[1]

## [1] 0.8632279
```

2.3 General correlated designs, `corDiGGer`

`corDiGGer` is a wrapper function to perform the standard searches of the previous `DiGGer` versions. Standard searches consisted of a blocking phase in the absence of correlation followed by a search phase with random row and column blocks and autoregressive correlation of 0.5 between rows and between columns.

The dimension pairs in the code that follows for the `blockSequence` list define sets of blocks for a search phase.

- `c(nr,nc)`
  - [`nr` × `columnsInDesign`]
  - [`rowsInDesign` × `nc`]
  - [`nr` × `nc`]

```
d18 <- corDiGGer(
  numberOfTreatments = 18,
  rowsInDesign = 18,
  columnsInDesign = 3,
  blockSequence = list(c(6,1)),
  rngSeeds = c(111, 222))
## plot(d18)
```

The `desTab` function calculates the frequencies of treatments occurring in blocks and shows that replicates have been set in two directions from this completely randomised initial design.

```
desTab(getDesign(d18), 18, 1)

##         B1 B2 B3
## freq_1 18 18 18
```

```
desTab(getDesign(d18), 6, 3)

##        B1 B2 B3
## freq_1 18 18 18
```

`corDiGGer` has a `blockSequence = 'default'` option which sets up blocks that cut across replicates. The following example of 20 treatments arranged in a [20×3] layout with [20×1] replicates shows the action when replication in two directions is not possible. Four equal blocks of [5×3] can be formed, equivalent to setting `blockSequence = list(c(5,1))`.

```
d20 <- corDiGGer(
    numberOfTreatments = 20,
    rowsInDesign = 20, columnsInDesign = 3,
    rowsInRep = 20, columnsInRep = 1,
    blockSequence = 'default',
    rngSeeds = c(111, 333))
```

```
getBlock(d20, phaseNo = 1, objectiveNo = 1,
         blockNo = 1)

## B RANDOM BLOCK: 5 Rows by 3 Columns, VarianceRatio 1.000000

getBlock(d20, 1,2,1)

## B RANDOM BLOCK: 5 Rows by 1 Column, VarianceRatio 1.000000

desTab(getDesign(d20), 5, 3)

##        B1 B2 B3 B4
## freq_1 15 15 15 15
```

The `treatRepPerRep` parameter allows for specification of unequal replication in replicate blocks. Consider a design for 33 entries where the first treatment is replicated 4 times with a [36×4] design and [36×1] replicate layout.

```
d33 <- corDiGGer(
    numberOfTreatments = 33,
    rowsInDesign = 36, columnsInDesign = 4,
    rowsInRep = 36, columnsInRep = 1,
    blockSeq = list(c(9,1)),
    treatRepPerRep= c(4,rep(1,32)),
    rngSeeds = c(111, 222))
## plot(d33)
## plot(d33, trts=1, col=2, new=FALSE)
```

```
desTab(getDesign(d33), 9, 4)

##         B1 B2 B3 B4
## freq_1 32 32 32 32
## freq_4  1  1  1  1

desTab(getDesign(d33), 9, 1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13
## freq_1  9  9  9  9  9  9  9  9  9   9   9   9   9
##         B14 B15 B16
## freq_1   9   9   9
```

The search has found replicates in two directions with each of the sixteen [9×1] blocks having no repeats of the higher frequency treatment.

2.4 Partially replicated designs, `prDiGGer`

Partially replicated designs have some treatments that are unreplicated and rely on replicated treatments to make the trial analysable. Partially replicated design were described in Cullis et al. (2006). It is recommended that at least 20% of the experimental units are occupied by replicated treatments. The aim of these experiments is usually to select promising treatments from a set of replicated and unreplicated test treatments, with check and quality standard treatments providing the necessary replication overall to give a valid experiment.

Group codes are used to distinguish between the different treatment types, standards and tests, with the default expectation that test treatments are coded as group 1.

`prDiGGer` proceeds by optimising a reduced design of replicated treatments using a scaled version of the overall block sequence. Once these blocks are established they are expanded to the size of the full design and augmented with unreplicated treatments as described in Herzberg and Jarrett (2007). The design has not been spatially optimised at this point.

The final step is to `run` the `DiGGer` object to spatially optimise the location of replicated treatments within the smallest blocks of the blocking sequence.

`pr293` is an unreplicated design with 288 unreplicated treatments and 5 standards each replicated 18 times. Approximately 24% of the experimental units will have replicated treatments. The block sequence has been chosen to match the replication level in the design. The [7×3] blocks result in 18 blocks spread across the design and should allow for each block to contain a set of the 5 standards. The sub-blocks of [7×1] will encourage replicated entries to be evenly placed within columns of the design.

```
pr293 <- prDiGGer(numberOfTreatments = 293, rowsInDesign = 21,
               columnsInDesign = 18,
               blockSequence = list(c(7,3), c(7,1)),
               treatRepPerRep = rep(c(1,18), c(288, 5)),
               treatGroup = rep(c(1, 2), c(288, 5)),
                  rngSeeds = c(156, 444))
## pr293
pr293 <- run(pr293)
## plot(pr293)
## plot(pr293, trts = 289:293, col = 2, new = FALSE)
```

```
desTab(getDesign(pr293), 7, 3)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13
## freq_1 21 21 21 21 21 21 21 21 21  21  21  21  21
##         B14 B15 B16 B17 B18
## freq_1  21  21  21  21  21
```

```
desTab(getDesign(pr293), 21, 1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13
## freq_1 21 21 21 21 21 21 21 21 21  21  21  21  21
##         B14 B15 B16 B17 B18
## freq_1  21  21  21  21  21
```

pr118 is a partially replicated design with 56 unreplicated test treatments, 56 test treatments with two replicates and 6 standards with 4 replicates. The layout [16×12] suggests a nested blocking sequence:

- [16×6] to give 2 blocks
- [8×6] to give 4 blocks
- [8×1] to optimise column blocks.

With `runSearch = TRUE` the final spatial optimisation is run immediately after the blocking phase augmentation.

```
pr118 <- prDiGGer(118, 16, 12,
                  blockSeq = list(c(16,6), c(8,6), c(8,1)),
                  treatRepPerRep = rep(c(1,2,4), c(56,56,6)),
                  treatGroup = rep(c(1,1,2), c(56,56,6)),
                  rngSeeds = c(8842, 9004),
                  runSearch = TRUE)
```

`desTab` is used to give a quick check of replication levels in each block.

```
desTab(getDesign(pr118), 16,6)

##         B1 B2
## freq_1 84 84
## freq_2  6  6

desTab(getDesign(pr118), 8,6)

##         B1 B2 B3 B4
## freq_1 48 48 48 48
```

```
desTab(getDesign(pr118), 8,1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13
## freq_1  8  8  8  8  8  8  8  8  8   8   8   8   8
##         B14 B15 B16 B17 B18 B19 B20 B21 B22 B23
## freq_1   8   8   8   8   8   8   8   8   8   8
##         B24
## freq_1   8
```

```
plot(pr118, trts=57:112, col=5, new=TRUE)
plot(pr118, trts=113:118, col=2, new=FALSE)
plot(pr118, trts=1:56, col=8, new=FALSE, label=FALSE)
```

2.5 Factorial designs `facDiGGer`

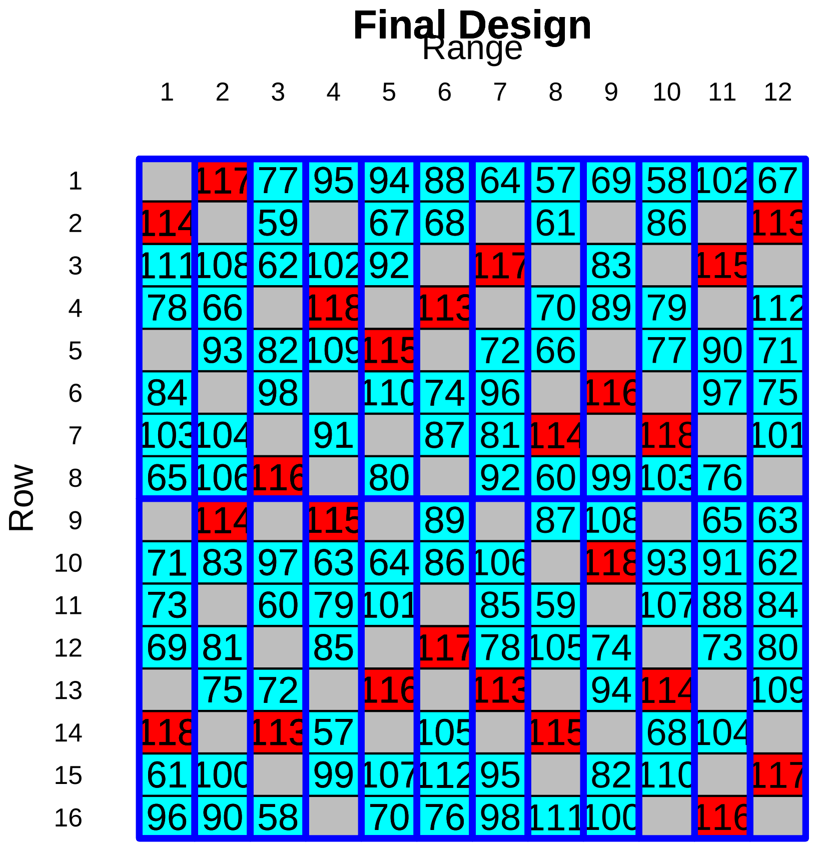
`facDiGGer` is a workaround to produce factorial designs using DiGGer. The function allocates treatment factors one at a time to the design, forming the interaction treatments at each step and using the last factor added as a group code in the search. `corDiGGer` is called once for each factor added.

After the first factor is allocated, the design for the levels of this factor is fixed. The levels of the second factor are allocated to each level of the first factor within a replicate. Treatment interchanges within replicates are restricted to interchanges between plots having the same first factor level. Before adding subsequent factors the design for the current interaction is fixed, with levels of the next factor allocated to each of the current interaction levels.

The function requires a treatment data frame with a column for each factor and the replication level of each factor combination.

A design template file may be used where there are missing plots in the rectangular design. The design template may have replicate codes which will be used to allocate replicates of the treatments from the treatment data frame. Missing plots are coded as `0`.

`createFactorialDF` is a utility function to create treatment data frames suitable for `facDiGGer`.



Row	1	2	3	4	5	6	7	8	9	10	11	12
1		117	77	95	94	88	64	57	69	58	102	67
2	114		59		67	68		61		86		113
3	111	108	62	102	92		117		83		115	
4	78	66		118		113		70	89	79		112
5		93	82	109	115		72	66		77	90	71
6	84		98		110	74	96		116		97	75
7	103	104		91		87	81	114		118		101
8	65	106	116		80		92	60	99	103	76	
9		114		115		89		87	108		65	63
10	71	83	97	63	64	86	106		118	93	91	62
11	73		60	79	101		85	59		107	88	84
12	69	81		85		117	78	105	74		73	80
13		75	72		116		113		94	114		109
14	118		113	57		105		115		68	104	
15	61	100		99	107	112	95		82	110		117
16	96	90	58		70	76	98	111	100		116	

Figure 1: Partially replicated design for 118 treatments

```
DF45 <- createFactorialDF(c(3,3,5))
head(DF45)

##   TRT F1 Fname1 F2 Fname2 F3 Fname3 Repeats
## 1   1  1     A1  1     B1  1     C1       1
## 2   2  1     A1  1     B1  2     C2       1
## 3   3  1     A1  1     B1  3     C3       1
## 4   4  1     A1  1     B1  4     C4       1
## 5   5  1     A1  1     B1  5     C5       1
## 6   6  1     A1  2     B2  1     C1       1
```

The `factorNames` order is the order in which factors are allocated. The allocation has better flexibility if the factors are allocated in the order of increasing number of levels.

```
d3x3x5 <- facDiGGer(
    factorNames = c("F1", "F2", "F3"),
    rowsInDesign = 45, columnsInDesign = 3,
    rowsInRep = 45, columnsInRep = 1,
    blockSequence = list(c(15,1)),
    treatDataFrame = DF45,
    treatRepColumn = "Repeats",
    maxInt=500000, rngSeeds = c(11283, 23951))
```

```
desTab(getDesign(d3x3x5),15,3)

##         B1 B2 B3
## freq_1 45 45 45

desTab(matrix(d3x3x5$dlist$F1,45),15,1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9
## freq_5  3  3  3  3  3  3  3  3  3

desTab(matrix(d3x3x5$dlist$F2,45),15,1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9
## freq_5  3  3  3  3  3  3  3  3  3
```

```
desTab(matrix(d3x3x5$dlist$F3,45),15,1)

##         B1 B2 B3 B4 B5 B6 B7 B8 B9
## freq_3   5  5  5  5  5  5  5  5  5
```

Tabulation shows replicates in the second direction and the [15 × 1] blocks have balanced allocation of main effect levels. If the factor order is changed so that the 5 level factor is not last the third factor allocated is not balanced within 15 unit blocks (not shown).

The example in Section 3.2 of Cochran and Cox (1957) is a factorial with added control with unequally replicated treatments.

```
DF9 <- createFactorialDF(
    c(2,3,5),
    treatName=list(Control = c("Control", "Treated"),
        Level = c(0,1,2),
        Fumigant = c("Nil", "CN", "CS", "CM", "CK")))
DF9 <- DF9[-c(2:15,16:20,21,26),]
DF9$Repeats <- rep(c(4,1), c(1,8))
xtable(DF9, caption = 'Data frame DF9', label = 'tab:df9a',
       include.rownames = FALSE, digits = 0)
```

	TRT	F1	Control	F2	Level	F3	Fumigant	Repeats
1	1	1	Control	1	0	1	Nil	4
22	22	2	Treated	2	1	2	CN	1
23	23	2	Treated	2	1	3	CS	1
24	24	2	Treated	2	1	4	CM	1
25	25	2	Treated	2	1	5	CK	1
27	27	2	Treated	3	2	2	CN	1
28	28	2	Treated	3	2	3	CS	1
29	29	2	Treated	3	2	4	CM	1
30	30	2	Treated	3	2	5	CK	1

Table 1: Data frame DF9

Table 1 shows the factor order and replication levels. There is an argument `chequerboard` used with two-level factors. When set `TRUE` the standard spatial model is used with two-level factors resulting in chequerboard self-diagonals of the levels. The `blockSequence = c(1,1)` is a special case which fits row and column blocks and spatial correlation in separate objectives.

```
test9x <- facDiGGer(factorNames = c("Control","Level","Fumigant"),
                    rowsInDesign = 6, columnsInDesign = 8,
                    rowsInRepl = 3, columnsInRep = 4,
                    blockSequence = list(c(1,1)),
                    chequerboard = TRUE,
                    treatDataFrame=DF9, rngSeeds = c(111,444))
```

```
desPlot(matrix(test9x$dlist$Level,6), 0, col=8,
         new=TRUE, label=FALSE)
desPlot(matrix(test9x$dlist$Level,6), 1, col=5,
         new=FALSE, label=FALSE)
desPlot(matrix(test9x$dlist$Level,6), 2, col=7,
         new=FALSE, label=FALSE)
desPlot(matrix(test9x$dlist$Fumigant,6),
         new=FALSE, label=TRUE, chtdiv=3,
         bdef=cbind(3,4), bcol=4, bwd=4)
```

Split-plot designs rely on settings of `mainPlotSizes` to define the nested main plot sizes in split-plot designs. After the split-plot search phases `mainPlotSizes` should be `c(1,1)`. Consider a trial with a 3-level factor in main plots and a 5-level factor on experimental units. The block structure of this experiment is:

- Design: [15×3]
- Replicate: [15×1]
- Main plot: [5×1]
- Sub-plot: [1×1]

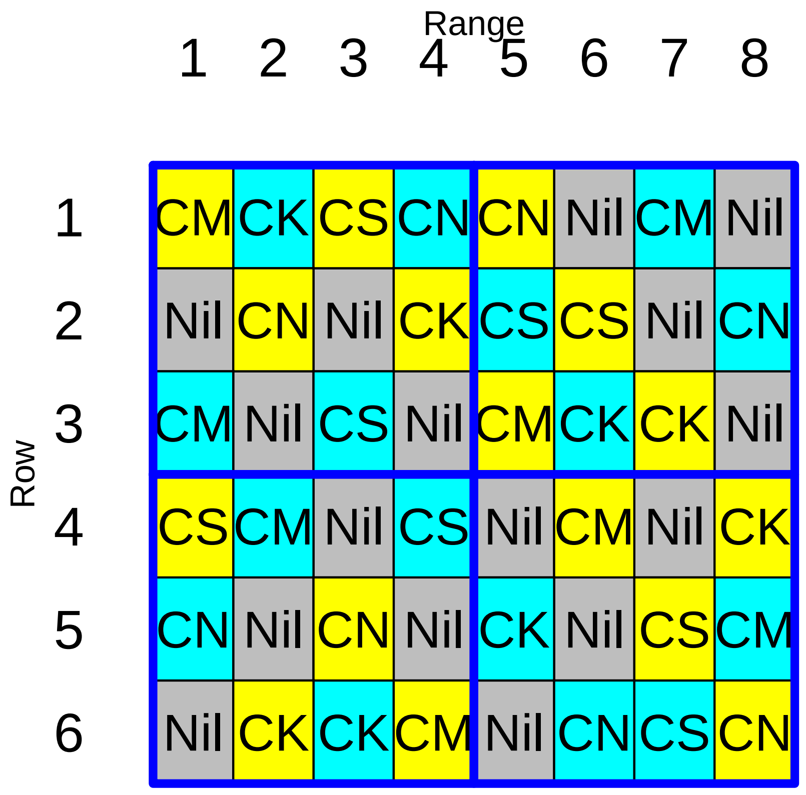

Range
1 2 3 4 5 6 7 8
Row
1 CM CK CS CN CN Nil CM Nil
2 Nil CN Nil CK CS CS Nil CN
3 CM Nil CS Nil CM CK CK Nil
4 CS CM Nil CS Nil CM Nil CK
5 CN Nil CN Nil CK Nil CS CM
6 Nil CK CK CM Nil CN CS CN

Figure 2: Factorial plus added control design.

```r
DF15 <- createFactorialDF(c(3,5))
sp3x5 <- facDiGGer(
  factorNames = c("F1", "F2"),
  rowsInDesign = 15, columnsInDesign = 3,
  rowsInRep = 15, columnsInRep = 1,
  mainPlotSizes = list(c(5,1), c(1,1)),
  treatDataFrame = DF15,
  treatRepColumn = "Repeats",
  maxInt=100000)
```

```r
plot(sp3x5, trts=1:5, col=5, new=TRUE, label=FALSE)
plot(sp3x5, trts=6:10, col=7, new=FALSE, label=FALSE)
plot(sp3x5, trts=11:15, col=8, new=FALSE, label=FALSE)
desPlot(matrix(sp3x5$dlist$F2,15), new=FALSE)
```

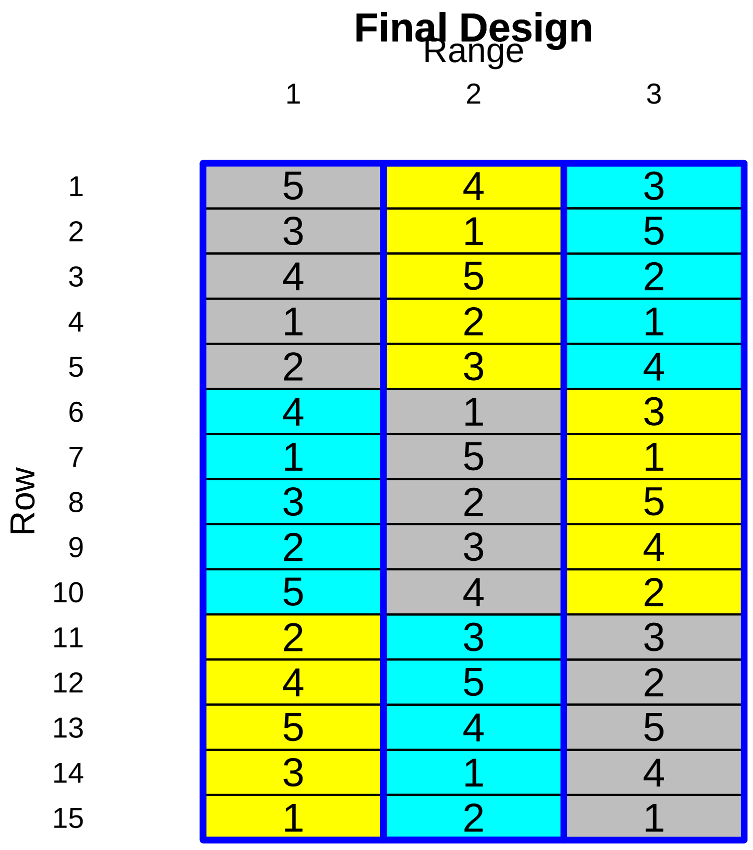
2.6 Strict 2D designs, `r2dDiGGer`

Strict replication in two dimensions was requested to ensure that any single treatment was spread in well separated zones where rectangular replicates were not possible. The restrictions on randomisation imposed by replication in two directions are that treatments are divided into sets that:

- always occur together in blocks
- occur with other sets in one block
- never occur together in blocks.

The 20 treatment example looked at in the `corDiGGer` section can be replicated in two directions using `r2dDiGGer`.

```r
d20a <- r2dDiGGer(numberOfTreatments = 20,
                  rowsInDesign = 20, columnsInDesign = 3,
                  rowsInRep = 20, columnsInRep = 1,
                  rngSeeds = c(111, 333))
```



Row	1	2	3
1	5	4	3
2	3	1	5
3	4	5	2
4	1	2	1
5	2	3	4
6	4	1	3
7	1	5	1
8	3	2	5
9	2	3	4
10	5	4	2
11	2	3	3
12	4	5	2
13	5	4	5
14	3	1	4
15	1	2	1

Figure 3: Split plot design for 3 main plot and 5 sub-plot treatments.

```
desPlot(matrix(rep(seq(3), each = 20), ncol = 3,
        byrow = TRUE), 1, col = 5,
        new = TRUE, label = FALSE)
desPlot(matrix(rep(seq(3), each = 20), ncol = 3,
        byrow = TRUE), 2, col = 7,
        new = FALSE, label = FALSE)
desPlot(matrix(rep(seq(3), each = 20), ncol = 3,
        byrow = TRUE), 3, col = 8,
        new = FALSE, label = FALSE)
desPlot(getDesign(d20a), new = FALSE,
         label = TRUE,
         bdef = cbind(20,1), bcol = 4, bwd = 4)
```

References

NSW DPI Biometrics software download page, 2009. URL `http://www.austatgen.org/files/software/downloads`.

W.G. Cochran and G.M. Cox. *Experimental Designs*. Wiley International Edition, New York, 2nd edition, 1957.

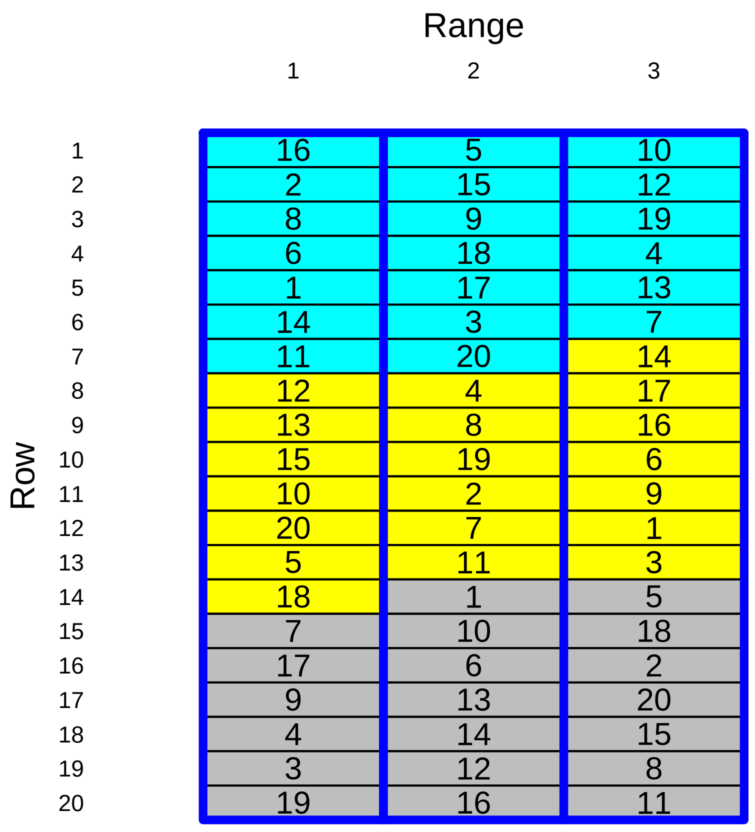
N.E. Coombes. *The Reactive Tabu Search for efficient correlated experimental designs*. PhD thesis, Liverpool John Moores University, 2002.

B.R. Cullis, A.B. Smith, and N.E. Coombes. On the design of early generation variety trials with correlated data. *Journal of Agricultural, Biological and Environmental Statistics*, 11:381–393, 2006.

A.M. Herzberg and R.G. Jarrett. A-optimal block designs with additional singly replicated treatments. *Journal of Applied Statistics*, 34:61–70, 2007.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

W.N. Venables and J.A. Eccleston. Randomized search strategies for finding near-optimal block or row-column designs. *Australian Journal of Statistics*, 35:371–382, 1993.



Row	1	2	3
1	16	5	10
2	2	15	12
3	8	9	19
4	6	18	4
5	1	17	13
6	14	3	7
7	11	20	14
8	12	4	17
9	13	8	16
10	15	19	6
11	10	2	9
12	20	7	1
13	5	11	3
14	18	1	5
15	7	10	18
16	17	6	2
17	9	13	20
18	4	14	15
19	3	12	8
20	19	16	11

Figure 4: Non-rectangular replicates in two directions, `d20a`.