DiGGer design search tool in R.

Neil Coombes

June 29, 2009

Contents

1 Introduction **1**

2 Preliminary concepts **2**
2.1 What is a design? . . . 2
2.2 Design properties . . . 3
2.2.1 Blocks . . . 3
2.2.2 BlockIn2D . . . 3
2.2.3 Correlation . . . 4
2.2.4 Efficiency of the design . . . 5
2.3 Searching for improvements . . . 5

3 The DiGGer() function **6**
3.1 The default search strategy. . . . 8
3.2 A simple call to DiGGer . . . 9
3.3 Checking the DiGGer object . . . 11

4 DiGGer() in detail **12**
4.1 Parameters in DiGGer() . . . 13
4.2 DiGGer objects . . . 16
4.3 Modifying the search . . . 18

5 Examples **20**
5.1 Standard Variety Trials . . . 20
5.1.1 Replicates in two directions . . . 20
5.1.2 Incomplete blocks across replicates . . . 20
5.2 Unequal replicate shapes . . . 21
5.3 Additional controls in a replicated design . . . 21
5.4 Functions using DiGGer . . . 23
5.4.1 Incomplete block designs . . . 24
5.4.2 Split-plot designs . . . 25
5.4.3 Strip-plot designs . . . 26
5.5 DiGGer for p-rep designs. . . . 27
5.5.1 Optimisation at a single site . . . 28
5.5.2 Additional controls in a replicated design revisited . . . 31
5.6 Embedded p-rep designs . . . 32

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).

The user specifies parameters for the number of treatments and the design layout in a call to `DiGGer()`. This sets up a search framework which may be viewed and modified before running the search using the `run()` method. `DiGGer` calls an external Fortran program (Coombes, 2002) to run the search. The command `library(DiGGer)` loads the `DiGGer` package and its required packages into R.

`DiGGer` provides object oriented methods, through the `R.oo` and the `R.methodsS3` packages, which allow complex searches to be formulated and executed. The `DiGGer()` function creates a `DiGGer` object which has component search `Phase`, `Objective`, `Correlation` and `Block` objects each with their own generating function.

This document describes the search process, gives an introduction to the `DiGGer` package and provides examples demonstrating some standard design situations. Parameters and `DiGGer` functions are shown in `typewritertext` font.

2 Preliminary concepts

2.1 What is a design?

A design is a matrix of design numbers. The design numbers are consecutive, 1 to `NumberOfTreatments`, at the appropriate replication. Each number represents a treatment applied to an experimental unit, or plot, in the trial or experiment. The matrix has `RowsInDesign` rows and `ColumnsInDesign` columns. Field experiments are usually physically arranged in rows and columns but many experiments can be notionally described in terms of rows and columns.

The design may be arranged in replicates in a Randomised Complete Block (RCB) design. Each replicate defined by [`RowsInReplicate` × `ColumnsInReplicate`] has treatments repeated according to `TreatmentRepeatsPerReplicate`.

`TreatmentRepeatsPerReplicate` is a vector `NumberOfTreatments` long with the relative replication levels for each treatment in a replicate.

k`*sum(TreatmentRepeatsPerReplicate)=RowsInReplicate*ColumnsInReplicate`,
k an integer. `DiGGer` will randomise k repeats of the replication scheme within each replicate.

The blocks with template [`RowsInReplicate`×`ColumnsInReplicate`] must tile evenly within the design (Figure 1). All blocks tile from Row 1, Column (or Range) 1.

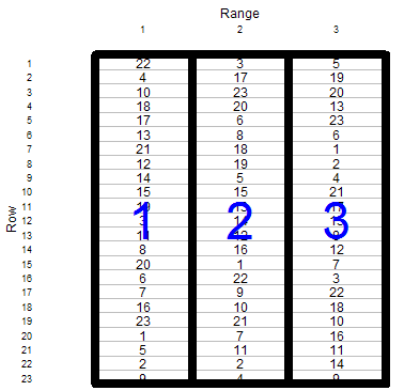


Figure 1: Tiling of replicates within a design.

If the design is completely randomised (CR) then

`RowsInReplicate=RowsInDesign`

`ColumnsInReplicate=ColumnsInDesign` and

k`*sum(TreatmentRepeatsPerReplicate)=RowsInDesign*ColumnsInDesign`,
k an integer. `DiGGer` will randomise k repeats of the replication scheme within the design.

2.2 Design properties

2.2.1 Blocks

The design may be divided into blocks. Blocks specify sections of the design with more similar experimental material. `Block` objects in `DiGGer` are defined with three parameters:

- `RowsInBlock`
- `ColumnsInBlock`
- `BlockVarianceRatio`.

If the `BlockVarianceRatio` > 0 then blocks are random, otherwise blocks are fixed. Blocks tile down and across the design from (Row 1, Column 1) using the block template, [`RowsInDesign` × `ColumnsInDesign`] (Figure 2). Blocks may tile unevenly:

- the final row of blocks may have fewer than `RowsInBlock` rows
- the final column of blocks may have fewer than `ColumnsInBlock` columns.

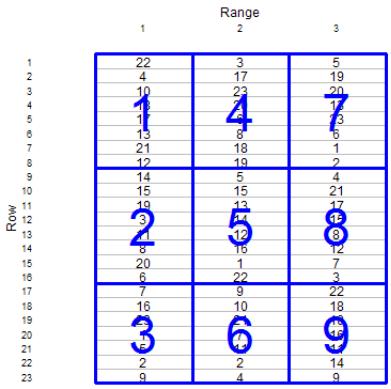


Figure 2: Tiling of blocks within a design. The final row of blocks (3,6,9) has one fewer row than the other blocks.

2.2.2 BlockIn2D

Blocking in two directions (Figure 3) is defined using a dimension pair `c(RowsInBlock,ColumnsInBlock)` which sets up the block templates

- RowBlocks with `RowsInBlock` rows: [`RowsInBlock` × `ColumnsInDesign`]
- ColumnBlocks with `ColumnsInBlock` columns: [`RowsInDesign` × `ColumnsInBlock`]
- RowBlocks.ColumnBlocks [`RowsInBlock` × `ColumnsInBlock`]

The latter is the set of intersection blocks induced by the former blocking structures and is important in finding the most efficient design.

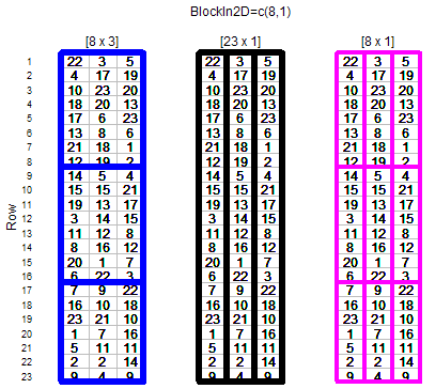


Figure 3: Blocking induced by `BlockIn2D=c(8,1)`: [8×3], [23×1] and [8×1] blocks.

Note that the set of intersection blocks is nested, or completely contained, within the other blocking structures. The relative importance of these blocking structures is controlled through `ObjectiveWeights` or `BlockVarianceRatios`.

2.2.3 Correlation

Correlation may be defined across the whole of the design or within independent blocks which must tile evenly into the design. There are three models for the correlation that may occur between plots within blocks: identical (`ID`), AutoRegressive (`AR`) and Moving Average (`MA`), .

- `ID`, equal correlation between plots within a block
- `AR`, the correlation ($\rho : -1.0 < \rho < 1.0$) between plots decreases with the separation of plots $\rho^{|i-j|}$, with i and j either row numbers or column numbers
- `MA`, adjacent experimental units are correlated ($\rho : -0.5 < \rho < 0.5$) but others are not correlated.

The correlation model assumes that the row correlation model and the column correlation model represent separable processes and the overall correlation is formed by the kronecker product of these correlations. For the `AR` ⊗ `AR` model treatment neighbours will tend to be balanced across rows and columns. The model ensures that the same treatment is strongly discouraged from occurring next to itself in rows or columns but it encourages a treatment to occur diagonally adjacent to itself as a self-diagonal pair. One reason to introduce blocking into the design is to reduce the possibility of self-diagonals.

Although it is not possible to know the true correlation between units in an experimental design the default AR model has been shown to be appropriate in many field designs. The designs produced are robust to misspecification of the correlation parameters. The designs produced for particular AR models are only marginally less efficient for AR models with different parameters.

2.2.4 Efficiency of the design

There are a number of ways of defining the efficiency of a design. The efficiency depends on the placement of treatments in the design, the blocks in the design and the assumed correlation across the design. `DiGGer` uses an A efficiency measure to compare candidate designs in a search. The A measure is the average variance of comparisons between pairs of treatments in the design. By minimising the A-measure `DiGGer` finds near optimal designs for given blocking and correlation structures.

Mathematically the information matrix, $\mathbf{C}$, of a design adjusted for nuisance fixed effects is

$$\begin{aligned}\mathbf{C} &= \mathbf{X}_1'\mathbf{\Lambda}^*\mathbf{X}_1 \\ \mathbf{\Lambda}^* &= \mathbf{\Lambda}^{-1} - \mathbf{\Lambda}^{-1}\mathbf{X}_2(\mathbf{X}_2'\mathbf{\Lambda}^{-1}\mathbf{X}_2)^{-1}\mathbf{X}_2'\mathbf{\Lambda}^{-1} \\ \mathbf{\Lambda} &= \sum \gamma_{b_i}\mathbf{Z}_{b_i}\mathbf{Z}_{b_i}' + \gamma_s\mathbf{\Sigma}_c \otimes \mathbf{\Sigma}_r\end{aligned}$$

where
$\mathbf{X}_1$ is the design matrix for the treatments
$\mathbf{X}_2$ is the design matrix for nuisance fixed effects such as the overall mean, fixed blocks and linear trend in rows or columns
$\mathbf{\Lambda}$ is the covariance matrix between plots of the design
$\mathbf{\Lambda}^*$ is the covariance matrix adjusted for nuisance fixed effects
$\mathbf{Z}_{b_i}$ is the design matrix for a random block factor with variance ratio γ_{b_i}
$\mathbf{\Sigma}_r$ is the spatial process between the rows
$\mathbf{\Sigma}_c$ is the spatial process between the columns
γ_s is the variance ratio associated with the spatial process.

The A value for comparing all treatments is proportional to the sum of the diagonal elements of the generalised inverse of the information matrix $\mathbf{C}^-$. The efficiency of the design is clearly dependent on the postulated blocking and correlation pattern in the design.

The efficiency of a design may also be calculated relative to particular treatment comparisons. Treatments may be divided into different groups of treatments and the efficiency may be based on within treatment group comparisons or between treatment group comparisons.

2.3 Searching for improvements

`DiGGer` is a search algorithm and requires

- an initial design
  - The initial design is a matrix of design numbers and is either generated from treatment replication and layout information or is supplied by the user.
- a method for changing the design
  - A design is modified by interchanging treatments that are applied to two experimental units.
  - Swap restrictions are required to preserve replicates and to preserve blocking structures that are created as part of the search. Swap restrictions are specified by `SwapBlock` dimensions and may also be restricted to interchanges between between experimental units with the same swap code in a user supplied `InitialSwap` matrix.
- an optimality/efficiency measure which depends on
  - the randomisation of the treatments
  - the correlation and blocking in the design
  - the treatments that are being compared.

The default optimality is the average variance of comparisons between all pairs of treatments and is given the code `A++` in `DiGGer`. It is 2/(`NumberOfTreatments` – 1) times the A-value of the design under the specified correlation and blocking. `DiGGer` aims to minimise the A-measure.

- rules to accept changes to the design
  - The algorithm in `DiGGer` is based on Reactive Tabu Search (RTS). It accepts any change which improves the best design in the search but allows the search to move to a poorer intermediate design when no improvement is possible from a single interchange.
  - A local minimum A-measure is reached when no treatment interchange improves the design. The search continues by choosing the best allowable interchange from a percentage of the possible interchanges as specified by the `SearchIntensity` property of a search phase. With the default `SearchIntensity=100`, the search for an improvement is concentrated close to the local minimum design. The lower the value of `SearchIntensity` the more randomness is introduced into the search.
- a method to stop the search
  - a predefined maximum number of interchanges has been tested
  - a target A-measure has been bettered.

The search itself may consist of a number of component searches, or search phases.

3 The `DiGGer()` function

The `DiGGer()` function was developed to produce a `DiGGer` R object which contains the framework of a search for a design which would be suitable for an agricultural field experiment. It can be thought of as a tool which has been set up for a typical job. Methods have also been created to enable the settings of the tool to be changed to suit a wide range of design situations. Once the design search object has been created it may be `run()` to produce a design.

The `DiGGer()` function has default settings:

```
NumberOfTreatments=18

RowsInDesign=18

ColumnsInDesign=3

RowsInReplicate=NULL

ColumnsInReplicate=NULL

BlockIn2D = TRUE

RowsInBlockSequence = NULL

ColumnsInBlockSequence = NULL

IndependentBlocks = NULL

Spatial = TRUE

RowColumn = TRUE

TreatmentType = "FIXED"

TreatmentVarianceRatio = -1
```

`TreatmentName = NULL`

`TreatmentNumber = NULL`

`TreatmentRepeatsPerReplicate = NULL`

`TreatmentGroup = NULL`

`TargetGroup = NULL`

`InitialDesign = NULL`

`InitialSwap = NULL`

`RNGSeeds = NULL`

The default search set up with a call `DiGGer()`

- generates a CR design with design numbers `sample(rep(seq(18),3))` in an [18×3] matrix
  - `NumberOfTreatments=18`
  - `RowsInDesign=18`
  - `ColumnsInDesign=3`
  - `TreatmentRepeatsPerRep=NULL`
  - `InitialDesign=NULL`
- sets up a component search (`Phase`) which optimises the design for [6×3], [18×1] and [6×1] blocks
  - `BlockIn2D=TRUE` (⇒ `BlockIn2D=c(6,1)`)
- sets up a component search (`Phase`) which optimises the new design for row ([1×3]), column ([18×1]) and error ([1×1]) random blocks in the presence of AR(0.5) correlation between rows and between columns
  - `RowColumn=TRUE`
  - `Spatial=TRUE`
- specifies that treatments are considered fixed effects in design
  - `TreatmentType="FIXED"`
  - `TreatmentVarianceRatio=-1`

The call to `DiGGer()` sets up a `DiGGer` search object which is an ordered collection of objects (Figure 4).

The CR initial design d_0 will be optimised in a search P_1 to an intermediate design d_1 which will in turn be optimised in a search P_2 to the final design d_2. The design d_1 will be optimised for [6×3], [18×1] and [6×1] blocks. The design d_2 will be optimised for AR spatial correlation and row and column random blocks while preserving blocks established in P_1. Each of the objects shown in Figure 4 may be queried and modified using appropriate methods. Methods of the form `getXXXX`, `setXXXX`, `addXXXX` and `deleteXXXX` exist for most objects `XXXX` defining the search.

Once the search has been set up the `run()` method runs the search. The FORTRAN `DiGGer` program is called from R and the progress of the search for each `Phase` is reported in 10% increments.

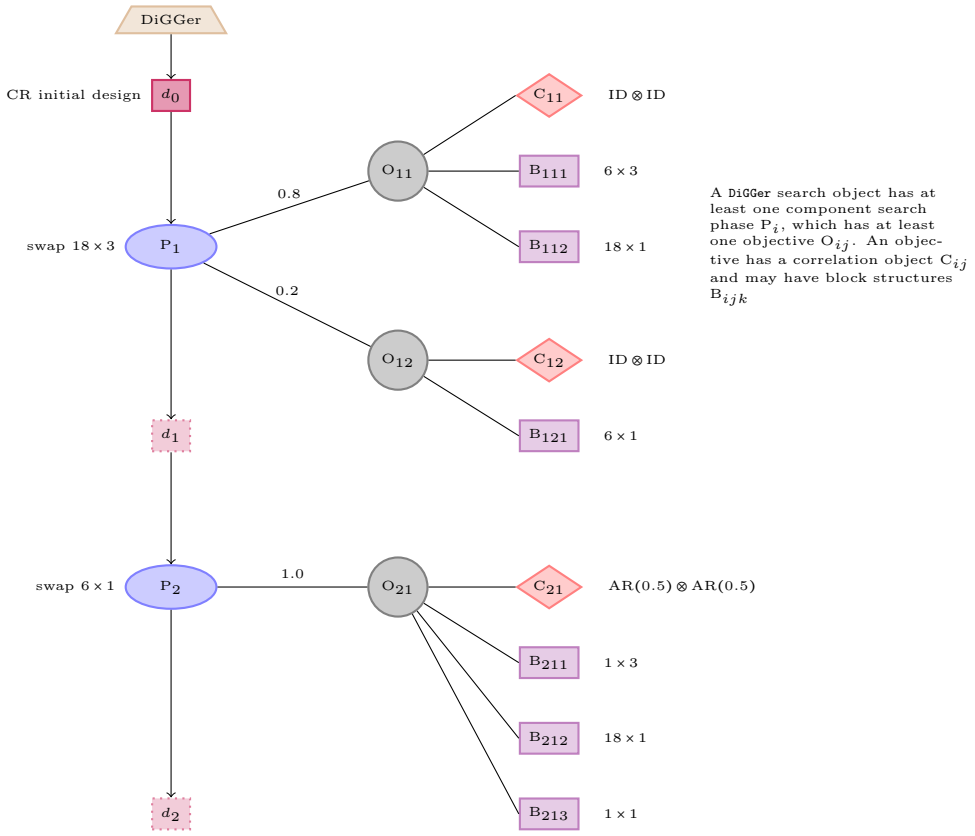


Figure 4: A `DiGGer` search as a collection of Phase, Objective, Correlation and Block objects.

3.1 The default search strategy.

The basic `DiGGer` search strategy is to

- Phase 1 search:
  - block a trial in two directions (`BlockIn2D=TRUE` or `BlockIn2D=c(b2r,b2c)`)
- Phase 2 search:
  - once those blocks are established, to arrange treatments within blocks to optimise a spatial objective (`Spatial=TRUE`, `RowColumn=TRUE`) across the whole design

If `BlockIn2D=TRUE` then

- a dimension not cut by replicates will be divided into three nearly equal blocks. Warning: `DiGGer` adds a dummy row or column for dimesions of size $3k+1$ so that template blocks of size $k+1$ tile to create 3 blocks $k+1 : k+1 : k$ with effective sizes $k : k+1 : k$.
- correlation is set to `ID` for both rows and columns
- the setting is ignored if replicates cut across both dimensions of the design

If `BlockIn2D=c(b2r,b2c)` then template blocks

- row blocks [`b2r` × `ColumnsInDesign`]
- column blocks [`RowsInDesign` × `b2c`]

- intersection blocks [`b2r` × `b2c`]

are optimised.

The final search phase allows both the specification of spatial correlation and row and column block effects in a single objective.

`Spatial=TRUE` creates a `Correlation` object

- `Correlation( RowModel="AR", RowParameter=0.5, ColumnModel="AR", ColumnParameter=0.5, SpatialVarianceRatio=1.0)`

`RowColumn=TRUE` creates 3 sets of blocks

- rows `Block(RowsInBlock=1, ColumnsInBlock=ColumnsInDesign, BlockVarianceRatio=1.0)`
- columns `Block(RowsInBlock=RowsInDesign, ColumnsInBlock=1, BlockVarianceRatio=0.5)`
- error `Block(RowsInBlock=1, ColumnsInBlock=1, BlockVarianceRatio=0.1)`

3.2 A simple call to `DiGGer`

Consider a design for 24 entries arranged [24 × 3] with replicates [24 × 1]. The simplest call to `DiGGer` sets up the search with

```
d24 <- DiGGer(24, 24,3, 24,1)
```

Note that the default order of arguments `NumberOfTreatments`, `RowsInDesign`, `ColumnsInDesign`, `RowsInReplicate` and `ColumnsInReplicate` is inferred and can reduce the amount of typing in R. The proposed search can be examined by typing `d24`.

```
INITIALISE:
Seeds -1 -1 -1 -1
Treatments 24 Fixed
Layout, Dsgn 24 x 3, Rep 24 x 1
P1 PHASE: Swap 24 x 1, Corr 24 x 3
          AType A++, Target 0.000000, MaxInt 10000
          Intensity 100.000000
  O11 OBJECTIVE: Weight 0.800000
                 Linear Covariate NONE
                 SPATIAL, Row ID(0.00) Col ID(0.00) Gamma 1.000000
                 B111 RANDOM BLOCK: 8 Rows by 3 Columns, VarianceRatio 1.0000
  O12 OBJECTIVE: Weight 0.200000
                 Linear Covariate NONE
                 SPATIAL, Row ID(0.00) Col ID(0.00) Gamma 1.000000
                 B121 RANDOM BLOCK: 8 Rows by 1 Column, VarianceRatio 1.0000
P2 PHASE: Swap 8 x 1, Corr 24 x 3
          AType A++, Target 0.000000, MaxInt 100000
          Intensity 100.000000
  O21 OBJECTIVE: Weight 1.000000
                 Linear Covariate NONE
                 SPATIAL, Row AR(0.50) Col AR(0.50) Gamma 1.000000
                 B211 RANDOM BLOCK: 1 Row by 3 Columns, VarianceRatio 1.0000
                 B212 RANDOM BLOCK: 24 Rows by 1 Column, VarianceRatio 0.5000
                 B213 RANDOM BLOCK: 1 Row by 1 Column, VarianceRatio 0.1000

Search not yet run.
```

The search is initialised with a generated RCB design (default setting `InitialDesign=NULL`).

Each `Phase` in `DiGGer` is a component search. The first phase (`P1`) is set up with two objectives (`O11` and `O12`) in response to the default `BlockIn2D=TRUE` parameter. Replicates do not cut across the 24 rows of the design so this dimension is divided into three equal blocks.
`P1`:

- allows interchanges in `SwapBlocks` [24×1] which correspond to replicates
- correlation and blocking are allowed to extend across the full design, `CorrelationBlock`: [24×3]
- objective `O11` defines the row blocking structure `B111` with block template [8×3] and `BlockVarianceRatio` set to 1.0. The A-value calculated for this uncorrelated objective is scaled by the `ObjectiveWeight` 0.8.
- objective `O12` defines the intersection blocks of `B111` with replicates. This uncorrelated blocking structure has block template [8×1] and `BlockVarianceRatio` set to 1.0. The A-value calculated for the design is scaled by the `ObjectiveWeight` 0.2.

The objective measure for a design is found by summing the A-value of each objective multiplied by its `ObjectiveWeight`. The default weights (0.8, 0.2) give more emphasis to the [8×3] blocks than the nested [8×1] blocks. As a consequence it is likely that the design after `Phase P1` will have replicates in [8×3] blocks. The replicate blocks [24×1] are retained because interchanges are only allowed in [24×1] blocks. Note that the weights required to get replicates in two directions depend on the number of treatments and block sizes. Weights may be changed as required using the `setObjectiveWeight` method.

The second search phase `P2` uses the design from `P1` as a starting design and is set up in response to the default `DiGGer` settings of `Spatial=TRUE` and `RowColumn=TRUE`.
`P2`:

- interchanges are now restricted to swaps between treatments within the blocks defined by the `SwapBlock` template [8×1]. This ensures that the efficient blocks established in `P1` are kept intact.
- correlation and blocking are allowed to extend across the full design, `CorrelationBlock`: [24×3]
- objective `O21` has the default spatial model with separable row and column `AR` autoregressive processes with parameters set at 0.5 together with random rows columns and error random blocks with `VarianceRatio`s of 1.0, 0.5 and 0.1 respectively.

The variance ratios were chosen to give more emphasis to rows because in standard field experiments replicates are defined in columns. If required the `Block` specifications may be changed using the `setBlock` method.

The search is run and the progress reported in 10% increments

```
> d24 <- run(d24)
     Phase,    Search%,    A-measure
[1] 1.0000000 0.0000000 0.6999907
[1]  1.0000000 10.0000000  0.6852174
[1]  1.0000000 20.0000000  0.6852174
[1]  1.0000000 30.0000000  0.6852174
[1]  1.0000000 40.0000000  0.6852174
[1]  1.0000000 50.0000000  0.6852174
[1]  1.0000000 60.0000000  0.6852174
[1]  1.0000000 70.0000000  0.6852174
```

```
[1]  1.0000000 80.0000000  0.6852174
[1]  1.0000000 90.0000000  0.6852174
[1]   1.0000000 100.0000000   0.6852174
 [1] 0.6666667 0.7594203 0.6852174 0.0000000 0.0000000 0.0000000 0.0000000
 [8] 0.0000000 0.0000000 0.0000000
8 rows by 1 column
     Phase,    Search%,    A-measure
[1] 2.0000000 0.0000000 0.5340626
[1]  2.0000000 10.0000000  0.4757147
[1]  2.0000000 20.0000000  0.4757147
[1]  2.0000000 30.0000000  0.4757147
[1]  2.0000000 40.0000000  0.4757147
[1]  2.0000000 50.0000000  0.4755902
[1]  2.0000000 60.0000000  0.4755902
[1]  2.0000000 70.0000000  0.4755902
[1]  2.0000000 80.0000000  0.4755902
[1]  2.0000000 90.0000000  0.4755902
[1]   2.0000000 100.0000000   0.4753482
 [1] 0.4753482 0.4753482 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 [8] 0.0000000 0.0000000 0.0000000
```

The `DiGGer` object `d24` now contains details of the completed search. Print methods are available to save the search to files or to R objects.

```
> View(d24$dlist)
> write.csv(d24$dlist,file="d24.csv",row.names=FALSE,quote=FALSE)
> print(d24,option="all",file="d24")
```

`d24$dlist` is a dataframe of the field-book listing of `d24`. The print call prints the set of files: `d24.in`, `d24.din`, `d24.swp`, `d24.trt`, `d24.sum` and `d24.lst`. These files could be used to run the standalone version of DiGGer.

3.3 Checking the `DiGGer` object

The results of a search should be checked carefully before being used in an experiment. There may be odd features in the design which render it unsuitable for practical use. Simple checks of a `DiGGer` object may include tabulation of treatments within blocks and plots of the design. The `plot()` method on a `DiGGer` objects produces the plot in Figure 5.

Treatment concurrences in blocks used in the design can be checked with the `getConcurrence()` method.

```
> getConcurrence(d24,phaseno=1,objectiveno=1,blockno=1)
B RANDOM BLOCK: 8 Rows by 3 Columns, VarianceRatio 1.000000

Treatment-Block

  3
276
> getConcurrence(d24,phaseno=1,objectiveno=2,blockno=1)
B RANDOM BLOCK: 8 Rows by 1 Column, VarianceRatio 1.000000

Treatment-Block

  0   1   3
 96 144  36
```

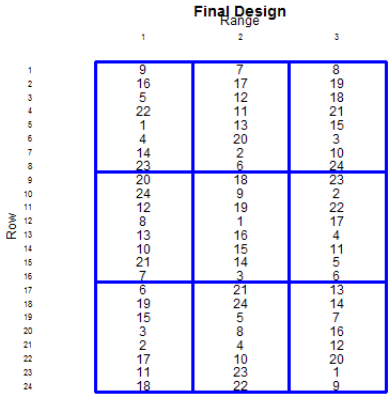


Figure 5: A default plot of `d24` produced with the command `plot(d24)`.

The first output of concurrences shows that all the possible 276 pairs of treatments occur together 3 times in [8×3] blocks and these blocks do form replicates. The second output of concurrences shows that 96 pairs of treatments do not occur together in [8×1] blocks, 144 pairs of treatments occur together once and 36 pairs of treatments occur together thrice. This is the optimal concurrence of treatments under the restriction that both the [24×1] and [8×3] blocks form replicates.

The matrix of design numbers can be extracted from the `DiGGer` object using the `getDesign()` method. `des.tab` is a function which tabulates the frequency of treatments occuring in blocks in a matrix. `des.plot()` is a function for plotting treatments within a matrix.

```
> m24 <- getDesign(d24)
> des.tab(m24,8,3)

 1
24

 1
24

 1
24
> des.plot(m24,m24[seq(8),1],col=5,new=TRUE)
> des.plot(m24,m24[seq(8)+8,1],col=6,new=FALSE)
> des.plot(m24,m24[seq(8)+16,1],col=7,new=FALSE,
+          bdef=cbind(8,1),bwd=4,bcol=3)
```

The tabulation of treatments in [8×3] blocks shows there is a replicate (24 treatments occurring once) in each of them. The plot of the design in Figure 6 shows that sets of 4 treatments occur together in three [8×1] blocks in the design.

4 `DiGGer()` in detail

This section looks in detail at the flexibility offered by the `DiGGer()` function and the methods available to modify the search to meet particular requirements. Section 4.1 describes the param-

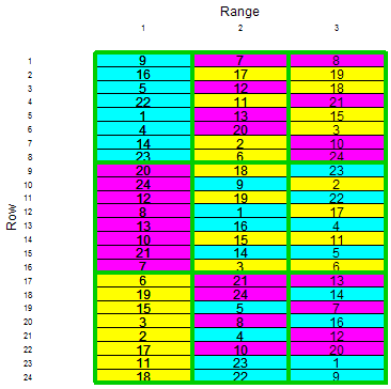


Figure 6: A plot to show the distribution of treatments from the first column of [8×1] blocks in the other two columns.

eters to the `DiGGer()` function. The function has been formulated to meet the requirements of common design situations from cereal breeding field experimentation. Section 4.3 shows how a design object may be queried and modified when the default search strategies do not meet the design's requirements.

4.1 Parameters in `DiGGer()`

Number of treatments:

- `NumberOfTreatments` is the number of distinct treatments in the design. The design `DiGGer` produces will have design numbers 1 to `NumberOfTreatments` at replication levels given in the `TreatmentRepeatsPerReplicate` parameter.
- The default setting `NumberOfTreatments=18` will usually need to be changed.

Design layout:

- the design is represented by a matrix with RowsInDesign rows and ColumnsInDesign columns
- each replicate has RowsInReplicate rows and ColumnsInReplicate columns. The template replicate block must tile evenly within the design. The `NULL` default settings of both parameters indicate the design is Completely Randomised (CR) with the replicate dimensions evaluated to the design dimensions.
- The default design layout is set to be CR in an [18×3] block.

Blocking in two directions:

- `BlockIn2D` sets up the first component search which optimises the treatment allocation to blocks in the absence of correlation.
  - `BlockIn2D=TRUE` causes `DiGGer` to divide the sets of rows and the sets of columns into three blocks each unless replicates cut across the dimension. The blocks are made as equal as possible. If `NumberOfRows` $\equiv 1 \pmod 3$ then a dummy row is added to the design. The dummy row has all design numbers set to zero and ensures that the first and final row blocks have only one fewer row than other row blocks. For example a design with `NumberOfRows` $= 16 \equiv 1 \pmod 3$ will have 17 rows in the augmented design

and three row blocks of size 6, 6 and 5 will be formed across the design by tiling a block with 6 rows. The design effectively has blocks of 5, 6 and 5 rows.

- `BlockIn2D=c(r,c)` forces blocks with [$r \times$ `ColumnsInDesign`], [`RowsInDesign` $\times c$] and [$r \times c$] to be optimised. The intersection blocks are placed in a separate objective and given a lower weight to reduce their importance relative to the larger blocking structures. If the design has more than 750 experimental units the intersection blocks are included with the other blocks in a single objective and their relative importance is scaled back with a lower block variance ratio.
- `BlockIn2D=c(1,1)` is a special case with only one component search. The objective formed has random rows and columns. If `Spatial=TRUE` then a separate spatial objective is added to this search.
- `BlockIn2D=FALSE` prevents blocking in two directions based on this setting. The blocking sequence parameters may still set up component searches to block in two directions.

Blocking sequences:

- the blocking sequence parameters are actioned after the `BlockIn2D` parameter. More complicated blocking features can be specified by two vectors of dimension specifications:
  - `RowsInBlockSequence=c(...)`
  - `ColumnsInBlockSequence=c(...)`

  where each pair of elements in the vectors is used in a `BlockIn2D`-type component search. An example of its use would be a quality trial where $n = 3p$ samples per day are processed over $m = 2k$ days in batches of p samples. Specifying
  - `RowsInBlockSequence=c(n,p,p)`
  - `ColumnsInBlockSequence=c(k,k,1)`

  sets up three component searches with blocks
  - [$n \times k$] dividing the experiment into `half` blocks
  - [$p \times m$] and [$p \times k$] dividing the experiment into `session` and `session.half` blocks
  - [$n \times 1$] and [$p \times 1$] dividing the experiment into `day` blocks and `session.day` blocks.

  `DiGGer` optimises each search in turn, adjusting the allowable swaps to maintain previously optimised blocks in the design.

Independent blocks:

- the default setting `IndependentBlocks=NULL` specifies that correlation and blocks may extend across the whole design
- when `IndependentBlocks=c(r,c)` the correlation is assumed to be within [$r \times c$] blocks only. Blocks are truncated to fit the correlation blocks. The correlation blocks must tile evenly within the full design. Truncation of correlation blocks is not allowed.
- independent blocks may be useful in glasshouse designs where blocks of pots are on separate benches. Correlation is specified between pots within benches but the benches are assumed to be independent with no correlation.

Controlling the final component search:

- `Spatial=TRUE` sets two correlation processes
  - between rows autoregressive (AR) correlation with parameter 0.5
  - between columns AR correlation with parameter 0.5

and sets the variance ratio for the correlated errors to be 1.0.

- `RowColumn=TRUE` sets three blocking factors
  - rows [$1 \times ColumnsInDesign$] with variance ratio 1.0
  - columns [$RowsInDesign \times 1$] with variance ratio 0.5
  - error [1×1] with variance ratio 0.1.

  The default variance ratios give more emphasis on row blocks because columns tend to be relatively efficient in common field trials which have one or two columns per replicate.

Treatment Information:

- the order that treatment information is provided is used as the design number in the design. There are four items of treatment information which may be added for each treatment in the design:
  - `TreatmentName` can supply a vector of texts with names for each treatment. These names will be used in the field-book listing of the trial. By default `TreatmentName=NULL` and dummy treatment names of the form `V_1` will be created.
  - `TreatmentNumber` can supply numeric identifiers for each treatment. The default `NULL` setting of this argument creates dummy identifiers $1 : $`NumberOfTreatments`. Treatment numbers are only used in the field-book listing of the design.
  - `TreatmentRepeatsPerReplicate` is a vector of replications for each of the treatments. The sum of the replications must equal the number of plots in a replicate. The default setting `TreatmentRepeatsPerReplicate=NULL` causes `DiGGer` to assume that all treatments are equally replicated in a replicate.
  - `TreatmentGroup` provides a code for each treatment which may be used to optimise particular treatment comparisons. The default action when `TreatmentGroup=NULL` is to set treatment group codes based on the replication level of the treatments. Treatments with the lowest replication are assigned code 1. If all treatment codes are 1 then the only optimisation available is the standard A measure comparing all treatment pairs.

A-measure types:

- There are seven types of A-measures available in `DiGGer`. All except `A++` require more than one treatment group. The A-measure types are:
  - `A++` measures the average pairwise variance of all treatment comparisons, $2(\text{trace}(\mathbf{C}^-) - \Sigma\mathbf{C}^-/t)/(t-1)$ where $\mathbf{C}^-$ is the inverse of the information matrix and t is the number of distinct treatments
  - `Agg` measures the average pairwise variance of group versus group treatment comparisons $t_i - t_j$ where the group codes $g_{t_i} \neq g_{t_j}$. `Agg` is particularly useful when blocking designs to ensure that treatments with different replication levels are evenly distributed to blocks.
  - `A22` measures the average pairwise variance of treatment comparisons between treatments coded with group code 2. This A-measure is often used in cereal breeding trials when test lines, unreplicated and replicated, have been coded with group code 2. The aim is to optimise comparisons between the treatments of interest.
  - `A11` measures the average pairwise variance of treatment comparisons between treatments coded with group code 1. As for `A22` this measure is often used where test lines of interest have been coded with `TreatmentGroup` code 1.
  - `A1+` measures the average pairwise variance of group 1 versus other group treatments, $t_i - t_j$ where the treatment t_i is a group code 1 treatment and treatment t_j is not a group code 1 treatment.

  - `Aa2` measures the average variance of comparisons between the average of group 1 treatments and treatments t_j where t_j is a group 2 treatment.
  - `Aaa` measures the variance of the comparison between the average of group 1 treatments and the average of group 2 treatments.

  The main A-measures used in `DiGGer` are: `A++` when treatments are equally replicated and there is no structure to the experiments, `Agg` during the blocking component search phases and `A22` or `A11` when there are some unreplicated test treatments of interest.

Target group:

- `TargetGroup` may be set to 1 or 2 for the final phase of the search. Blocking phases will normally use `Agg` as the optimality measure and the final phase may be set to have the optimality based on the target group, `A11` or `A22`. By default `TargetGroup=NULL` and `A++` is used for equally replicated treatments and `Agg` is used for unequally replicated treatments.

Initial design:

- the `InitialDesign` may be provided as a matrix of design numbers. Recall that design numbers correspond to the order in which treatment information is provided. The initial design must contain digits in the range 1 : `NumberOfTreatments`. The replication in the initial design must match the information provided by `TreatmentRepeatsPerReplicate`.
- if `InitialDesign=NULL` then `DiGGer` generates an initial design based on information in `TreatmentRepeatsPerReplicate` and the layout parameters, `RowsInDesign`, `ColumnsInDesign`, `RowsInReplicate` and `ColumnsInReplicate`. Note that if there are missing plots in the design the initial design must be supplied with design number 0 corresponding to missing plots in the design.

Initial swap matrix:

- the default setting `InitialSwap=NULL` allows treatment swaps between all pairs of plots subject to the definition of replicate layout.
- if `InitialSwap` is supplied it is a matrix of swap codes. Swaps are only allowed between units with the same swap code. Swap codes are required if replicates are not a tiled shape within the design or if there are treatments which must be fixed at specific locations in the design.

Random number generator seeds:

- by default `RNGSeeds=NULL` and `DiGGer` generates seeds which control the randomisation of the initial design when it is generated and the randomisation of the search when it is run. Users may supply a vector of four seeds for the random number generator `RNGSeeds=c(s1,s2,s3,s4)` where $0 \leq$ `s1`, `s3` ≤ 31328 and $0 \leq$ `s2`, `s4` ≤ 30081. `s1` and `s2` determine the generated initial design. `s3` and `s4` determine the swap ordering in the first search phase.

4.2 `DiGGer` objects

The call to `DiGGer()` creates a `DiGGer` object which is a collection of `Phase`, `Objective`, `Correlation` and `Block` objects. The hierarchy of these objects is:

- `DiGGer` object
  - `Phase` component search object
    - `Objective` object
      - `Correlation` object
      - `Block` object

There must be at least one `Phase`. Each `Phase` must have at least one `Objective` and the `Correlation` must be specified. An `Objective` may have no `Block` objects.

The `DiGGer()` function initialises the search by providing

- Treatment information
  - `NumberOfTreatments`
  - `TreatmentType`
  - `TreatmentVarianceRatio`
  - `TreatmentName`
  - `TreatmentNumber`
  - `TreatmentRepeatsPerReplicate`
  - `TreatmentGroup`
- Layout information
  - `RowsInDesign`
  - `ColumnsInDesign`
  - `RowsInReplicate`
  - `ColumnsInReplicate`
- Initial matrices
  - `InitialDesign`
  - `InitialSwap`

The `Phase` component search has the properties

- `SwapBlock: RowsInSwapBlock, ColumnsInSwapBlock` set to
  - replicate size in the initial search
  - the smallest established block from the last completed search phase
- `CorrelationBlock: RowsInCorrelationBlock, ColumnsInCorrelationBlock` set to the design dimensions.
- `AType` set to
  - `"A++"` for equally replicated treatments
  - `"Agg"` for unequally replicated treatments
  - `"A11"` or `"A22"` in the final phase if `TargetGroup` is specified as 1 or 2 respectively in `DiGGer()`.
- `ATargetValue` set to 0.0. The search phase is terminated when the A-measure falls below this value. Since A-measures are positive the default setting must be changed to be used to terminate a search phase.
- `MaxInterchanges` set to 10000 for all but the final search phase where it is set to 100000. This is the default stopping rule for a search phase. The search stops once `MaxInterchanges` treatment interchanges have been tested.
- `SearchIntensity` affects the randomness of the search. Set by default to 100 it is the percentage of the possible interchanges considered when any single interchange cannot improve the design. In some instances of block designs with no correlation better designs may be found with a low setting. Set to 0 the first interchange in the randomised list of possible swaps is accepted. The default setting is usually adequate.

- `NumberOfObjectives` is the number of `Objective` objects in this search phase.

The `Objective` object has properties

- `ObjectiveWeight` is the multiplier used to scale the A-measure of this objective. A single A-measure for multiple objectives is obtained by summing scaled A-measures. The relative importance of objectives may be controlled with this parameter. By default when blocking in 2 directions using two objectives the first objective has `ObjectiveWeight=0.8` and the intersection blocks are in an objective with `ObjectiveWeight=0.2`. These weights may need to be changed to get replicates in two directions depending on the `NumberOfTreatments`.
- `Correlation` gives the parameters of the `Correlation` object.
- `LinearCovariate` signifies linear trends present in the design and has settings
  - `"ROWS"` to fit a covariate for row number
  - `"COLUMNS"` to fit a covariate for column number
  - `"BOTH"` to fit both row and column covariates
  - `"NONE"` the default for no linear trends
- `NumberOfBlocks` gives the number of `Block` factors in this objective.

The `Correlation` object has settings

- `RowModel="ID", RowParameter=0.0,`
  `ColumnModel="ID", ColumnParameter=0.0,`
  `SpatialVarianceRatio=1.0` for blocking phases of the search
- `RowModel="AR", RowParameter=0.5,`
  `ColumnModel="AR",ColumnParameter=0.5,`
  `SpatialVarianceRatio=1.0` with `Spatial=TRUE`
- the model `"ID"` has parameter 0.0, `"AR"` has parameters $-1.0 < \rho < 1.0$ and `"MA"` $-0.5 < \rho < 0.5$.

The `Block` object has properties

- `RowsInBlock`, `ColumnsInBlock` and `BlockVarianceRatio`.
- if `BlockVarianceRatio`> 0 blocks are random, otherwise blocks are fixed. Fixed blocks are not recommended as they can cause problems in the search. If sets of blocks contain disjoint sets of treatments the design is disconnected and the search crashes. Random blocks avoid this problem. Note that as the `BlockVarianceMatrix`$\rightarrow \infty$ the random blocks approach fixed blocks.

4.3 Modifying the search

The `DiGGer` package aims to provide a flexible design tool which enables the user to formulate a search to find designs suited to a wide range of problems.

Typing the name of a `DiGGer` object prints the search to the computer screen. The search should be checked before being run. The `DiGGer` search object is a collection of component objects which may be modified. Methods exist to modify most aspects of a `DiGGer` search. The object may specified with identifiers `phaseno`, `objectiveno` and `blockno` where needed. Some examples are given here but more will be illustrated in the examples in Section 5. The full set of functions to change the search formulation are given in the `DiGGer` package documentation in R.

Suppose that two replicates of 24 wheat samples are milled sequentially in batches of 12 over 4 days. This design may be described with the layout $[\ 12 \times 4\ ]$. It is known that milling properties change over a day as the mill equipment heats up. It would be reasonable assume there are strong

correlations between samples processed close together within a day but there is no correlation between samples processed on different days. A call to `DiGGer` to set up a search with rows as samples per day and columns as days could be

```
> d24mill <- DiGGer(24, 12,4, 12,2, BlockIn2D=c(4,1), Spatial=TRUE,
+                   RowColumn=FALSE)
> d24mill
```

There are two phases to this search

- Phase 1: establishing blocks
  - `SwapBlock`: [12×2]
  - `Blocks`: `B111` [4×4], `B112` [12×1], `B121` [4×1]
- Phase 2: defining correlation across the blocked design
  - `SwapBlock`: [4×1]
  - `Correlation`: $\mathrm{AR}_c(0.5) \otimes \mathrm{AR}_r(0.5)$

The default `Spatial=TRUE` parameter in `DiGGer` sets up correlation across the design as AR with parameter 0.5 between rows and between columns. The correlation specified in a `DiGGer` search may be changed using the `setCorrelation` method.

The correlation in a `DiGGer` object may be queried with:

```
> getCorrelation(d24mill,phaseno=2,objectiveno=1)
SPATIAL, Row AR(0.50) Col AR(0.50) Gamma 1.000000
```

It may be reasonable to assume that there is correlation between samples within a day (between rows) but not between samples across days (between columns). The correlation may be set and the final search phase checked:

```
> setCorrelation(d24mill, phaseno=2, objectiveno=1,
+                RowModel="AR", RowParameter=0.7,
+                ColumnModel="ID", ColumnParameter=0.0,
+                SpatialVarianceRatio=1.0)
> getPhase(d24mill, 2)
P PHASE: Swap 4 x 1, Corr 12 x 4
          AType A++, Target 0.000000, MaxInt 100000
          Intensity 100.000000
  O1 OBJECTIVE: Weight 1.000000
                Linear Covariate NONE
                SPATIAL, Row AR(0.70) Col ID(0.00) Gamma 1.000000
                NO BLOCKS
```

`setCorrelation` changes the settings of the `Correlation` object in the second phase of the `DiGGer` object.

The number of interchanges required to optimise block objectives in a search is often much fewer than the 10000 default. In an earlier run of this job the first phase of the search had an optimum A-measure of 1.146249 and required 241 interchanges.The number of interchanges to be tested and the target A-measure may be set using

```
setMaxInterchanges(d24mill,phaseno=1,2000)
setTargetAValue(d24mill,phaseno=1,1.146250)
```

5 Examples

5.1 Standard Variety Trials

Standard variety trials in cereal breeding programs are often set out in 3 replicates in 3 or 6 columns. When the number of rows in the design is divisible by the number of replicates the design can be set out with replicates in two directions. Caution: this is a restriction on randomisation which necessarily results in sets of treatments always occurring together in blocks. The call to `DiGGer` is simple because the design tool defaults have these trials in mind.

5.1.1 Replicates in two directions

A design for 24 varieties arranged in 24 rows by 3 columns with replicates 24 rows by 1 column is found with:

```
> d24b <- DiGGer(24, 24,3, 24,1)
> d24b <- run(d24b)
```

The design produced can be viewed and saved to file

```
> plot(d24b)
> View(d24b$dlist)
> write.csv(d24b$dlist,file="d24b.csv",row.names=FALSE,quote=FALSE)
```

Two blocking structures are established in the first component search: [8×3] and [8×1]. The concurrences of pairs of treatments in these blocks can be checked:

```
> getConcurrence(d24b,phaseno=1,objectiveno=1,blockno=1)
B RANDOM BLOCK: 8 Rows by 3 Columns, VarianceRatio 1.000000

Treatment-Block

  3
276
> getConcurrence(d24b,1,2,1)
B RANDOM BLOCK: 8 Rows by 1 Column, VarianceRatio 1.000000

Treatment-Block

  0   1   3
 96 144  36
```

The blocks of [8×3] contain replicates and there are 36 pairs of treatments which occur together in three [8×1] blocks.

5.1.2 Incomplete blocks across replicates

If there are 32 varieties arranged in an experiment [16×6] with replicates [16×2] additional blocking across the replicates may be specified with the `BlockIn2D` parameter:

```
> d32ib <- DiGGer(32, 16,6, 16,2, BlockIn2D=c(4,2))
> d32ib <- run(d32ib)
 ...
> des.tab(getDesign(d32ib),4,6)
 1
24
```

```
 1
24

 1
24

 1
24
```

`BlockIn2D=c(4,2)` produces two blocking structures, [4×6] and [4×2], with the former structure containing 24 distinct treatments.

```
> getConcurrence(d32ib,1,1,1)
B RANDOM BLOCK: 4 Rows by 6 Columns, VarianceRatio 1.000000

Treatment-Block

  2   3
384 112
> getConcurrence(d32ib,1,2,1)
B RANDOM BLOCK: 4 Rows by 2 Columns, VarianceRatio 1.000000

Treatment-Block

  0   1   2   3
236 193  58   9

> write.csv(d32ib$dlist,file="d32ib.csv",row.names=FALSE,quote=FALSE)
```

5.2 Unequal replicate shapes

A design for 23 treatments and three replicates can be set out in a [9×8] array. Replicates of the treatments can be kept in contiguous but differently shaped replicates by creating an initial swap matrix coding the replicates and by creating an initial design with a randomisation of the treatments in these replicates. The last three experimental units will be missing plots coded with design number 0.

Create a matrix with swap codes, an initial design and set up the search

```
> s23 <- matrix(rep(c(1,2,3,0),c(23,23,23,3)),9)
> m23 <- matrix(c(sample(23),sample(23),sample(23),rep(0,3)),9)
> d23 <- DiGGer(23, 9,8,
                InitialDesign=m23,InitialSwap=s23)
```

The swap matrix illustrating the unequal replicates is shown in Figure 7. The call to `DiGGer`, by default, sets up `BlockIn2D=c(3,3)` forcing some balance to treatments occupying [3×8] and [9×3] blocks.

5.3 Additional controls in a replicated design

A design for 15 treatments has a control treatment replicated 4 times in each replicate. The design is set out [18×4] with corner replicates [9×2]. The standard call to `DiGGer` with unequally replicated treatments uses `AType="Agg"`. Here the AType is modified using `setAType(d15cc,1,"A++")` to show the effect of extra replication of controls in a replicated trial.

```
> trep <- rep(c(1,4),c(14,1))
> d15cc <- DiGGer(15, 18,4, 9,2,
```

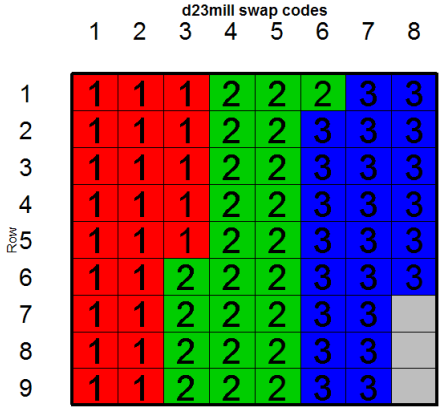


Figure 7: Swap codes for 23 treatments in unequally shaped replicates.

```
>                 TreatmentRep=trep)
> getAType(d15cc)
[1] "Agg"
> setAType(d15cc,1,"A++")
> d15cc <- run(d15cc)
```

The plot of the resulting design shows that the an A-measure based on the average variance of all pairwise comparisons pushes the controls to the perimeter of the trial (Figure 8).

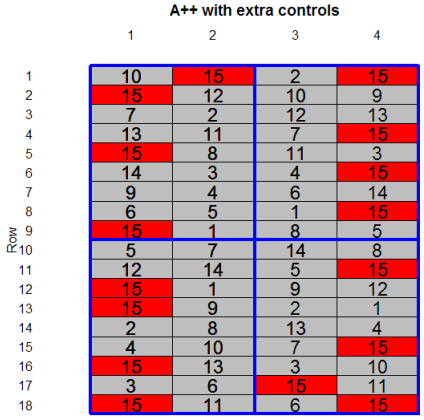
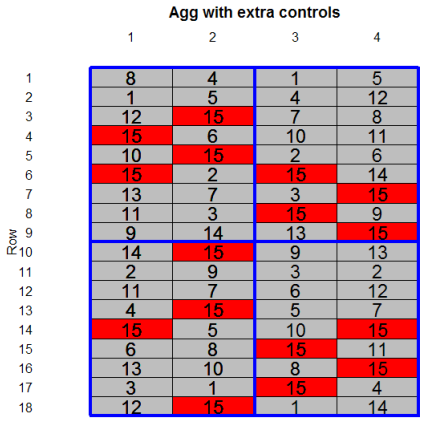


Figure 8: Distribution of 4 controls per replicate with A++ optimisation.

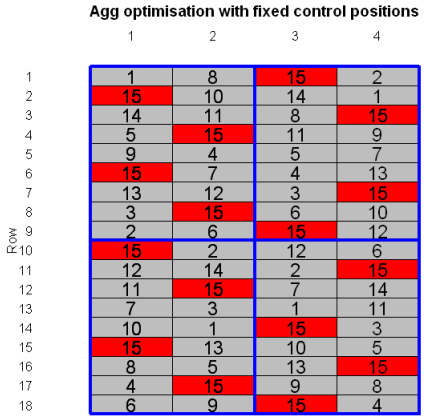
If the optimality measure is left at the default `"Agg"` then the controls are spread more evenly across the design (Figure 9). The control treatments, however, show the tendency to arrange themselves in self-diagonals. The available A-measures may not provide the most appropriate measure of optimality for the replication levels present in the design. An alternative approach considers the placement of the higher replicated treatment as a sampling exercise. What place-

ments of treatment 15 gives most information about the other plots in the design? The design of unreplicated experiments is described in Section 5.5. The placement of treatment 15 from an unreplicated design with the remaining treatments optimised around these fixed treatments is shown in Figure 10. A more complete discussion of this approach is given in Section 5.5.2.



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

Figure 9: Distribution of 4 controls per replicate with Agg optimisation.



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

Figure 10: Control treatment 15 from an optimised unreplicated design kept in fixed positions for an optimised replicated design.

5.4 Functions using `DiGGer`

Functions which call `DiGGer()` may be written to simplify the generation of scripts for specific design types. The function may call `DiGGer` a number of times to set up and make modifications to the standard search set up by `DiGGer`.

5.4.1 Incomplete block designs

Incomplete block designs have a single blocking structure and near optimal designs may be found using `DiGGer`. The `SearchIntensity` parameter is important here. With incomplete block designs there are many designs clustered near local minima in the search space. Setting the `SearchIntensity` to a low value introduces randomness to the search and may enable a route to the optimal design to be found.

The function `des.ib01` has the following form:

```
des.ib01 <- function(NumberOfTreatments,
                     RowsInDesign,
                     ColumnsInDesign,
                     RowsInReplicate=NULL,
                     ColumnsInReplicate=NULL,
                     RowsInBlock,
                     ColumnsInBlock,
                     FixedBlocks=FALSE,MaxInterchanges=100000,
                     SearchIntensity=100,
                     TargetAValue=0.0,
                     runSearch=TRUE,
                     ...)
```

The function will optimise the incomplete block design with blocks defind as `(RowsInBlock,ColumnsInBlock)` and changes the settings of the standard call to `DiGGer` to ensure the `Objective` has one block structure and no correlation. The user has the option of changing `MaxInterchanges` and `SearchIntensity` in the function call and setting the `TargetAValue` when the efficient value is known.

Suppose a design is required for 8 replicates of 15 treatments in blocks of 8, the call to `des.ib01` is:

```
> dib15k8 <- des.ib01(15, 15,8, RowsInBlock=1, ColumnsInBlock=8,
+                      SearchIntensity=100, MaxInt=200000)
     Phase,    Search%,    A-measure
[1] 1.000000 0.000000 0.278359
[1]  1.0000000 10.0000000  0.2648680
[1]  1.0000000 20.0000000  0.2648041
...
[1]   1.0000000 100.0000000   0.2648041
> getConcurrence(dib15k)
B RANDOM BLOCK: 1 Row by 8 Columns, VarianceRatio 1.000000

Treatment-Block

 3  4  5
 6 93  6
```

With `SearchIntensity=100` the treatment pair concurrences show some imbalance with 93 treatment pairs occurring together 4 times in blocks, 6 tratment pairs occurring together 3 times and 6 treatment pairs occurring together 5 times in blocks. If `SearchIntensity=0` is used the search uses a random interchange to escape from a local minimum in the search space. Recall that a local minimum occurs when no single interchange of treatments improves the current A-value.

```
> dib15k <- des.ib01(15, 15,8, RowsInBlock=1, ColumnsInBlock=8,
+                     SearchIntensity=0, MaxInt=200000)
     Phase,    Search%,    A-measure
[1] 1.0000000 0.0000000 0.2812088
```

```
[1]  1.0000000 10.0000000  0.2649007
[1]  1.0000000 20.0000000  0.2649007
[1]  1.0000000 30.0000000  0.2648526
[1]  1.0000000 40.0000000  0.2648526
[1]  1.0000000 50.0000000  0.2648043
[1]  1.0000000 60.0000000  0.2647059
...
[1]   1.0000000 100.0000000   0.2647059
> getConcurrence(dib15k)
B RANDOM BLOCK: 1 Row by 8 Columns, VarianceRatio 1.000000

Treatment-Block

  4
105
```

The increased randomness in the search has allowed the optimal block design with all pairs of treatments occurring together 4 times in blocks to be found. Taking a random interchange at a local minimum allows the current design to be moved well away from the local minimum giving the search an opportunity to avoid being trapped at a local minimum from which there is no route to the global minimum.

In searching for optimal block designs it is often better to repeat searches with fewer `MaxInterchanges` than to conduct a single search with a large number of interchanges.

5.4.2 Split-plot designs

Split-plot designs in a field setting have main plot treatments applied to contiguous sets of plots to which sub-plot treatments are applied. The function `des.split00` uses two calls to `DiGGer` to

- optimise the design for main plots
- randomise treatments within main plots to give the most efficient design for the interaction treatments.

A draft version of this function has the form

```
des.split00 <- function(nmaintrt=8,
                        nsubtrt=5,
                        RowsInDesign=20,
                        ColumnsInDesign=6,
                        RowsInReplicate=20,
                        ColumnsInReplicate=2,
                        RowsInMainPlot=5,
                        ColumnsInMainPlot=1,
                        continue=FALSE,
                        MainB2D=FALSE,
                        SubB2D=FALSE,
                        TreatmentName=NULL,
                        split.name.char='_',
                        ...)
```

and defines the layout of the experiment with

- Design shape: `RowsInDesign`, `ColumnsInDesign`
- Replicate shape: `RowsInReplicate`, `ColumnsInReplicate`
- Main plot shape: `RowsInMainPlot`, `ColumnsInMainPlot`.

When treatment names are supplied they are expected as

- a list with two elements
  - a text vector of main treatment names
  - a text vector of sub-treatment names
- or a text vector of names with sub-treatments nested within main treatments.

Consider a seeding rate trial where 6 variety main treatments will have 3 seeding rate sub-treatments applied. The design is [9×6] with replicates [9×2] and main plots shaped [3×1]. A split-plot design may be generated with

```
> vname <- c("Variety1","Variety2","Variety3","Variety4","Variety5","Variety6")
> dname <- c("D24","D36","D48")
> dsvd <- des.split00(nmaintrt=6, nsubtrt=3,
+                     RowsInDesign=9, ColumnsInDesign=6,
+                     RowsInReplicate=9, ColumnsInReplicate=6,
+                     RowsInMainPlot=3,ColumnsInMainPlot=1,
+                     MainB2D=c(1,2),
+                     TreatmentName=list(vname,dname))
...
> write.csv(dsvd$dlist,file="VarietyDensitySplit.csv",
+           row.names=FALSE,quote=FALSE)
> msvd <- getDesign(dsvd)
> msv <- dsvd$iswap
> msd <- matrix(rep(seq(3),6)[msvd],9)
> vcol <- rainbow(6)
> des.plot(msvd,label=FALSE)
> for (i in seq(6))
+     des.plot(msv,i,col=vcol[i],new=FALSE,label=FALSE)
+
> des.plot(msd,new=FALSE,bdef=cbind(3,2),bwd=3)
```

The code shows the specification of treatment names as a list. `MainB2D=c(1,2)` sets blocking in two directions so that the [1×2] blocks of main plots are considered in the optimisation. It prevents pairs of main plot treatments always occurring together in replicates. The final section of code plots the design. The matrix `dsvd$iswap` is the `InitialSwap` matrix created for the second call to `DiGGer` to restrict treatment swaps to within main plots. Main plots are coloured in the plot and subplot treatments are labelled (Figure 11).

5.4.3 Strip-plot designs

Strip-plot designs are described here in the field setting where treatments are applied in strips across replicates. The typical case is in row spacing experiments where machinery must be set to deliver seed at particular row spacings and the same row spacing must extend across all columns of the experiment. A draft function to produce split-plot designs `des.strip00` has the form:

```
des.strip00 <- function(nstrip.trt,         # no. of strip treatments
                        nother.trt,         # no. of other treatments
                        nrd,                # RowsInDesign
                        ncd,                # ColumnsInDesign
                        nrr,                # RowsInReplicate
                        ncr,                # ColumnsInReplicate
                        rows.in.strip=1,    # rows in each strip treatment
                        rows.in.block=NULL, # rows per block of strip treats
                        b2d=FALSE,          # BlockIn2D for interaction
             ...)
```

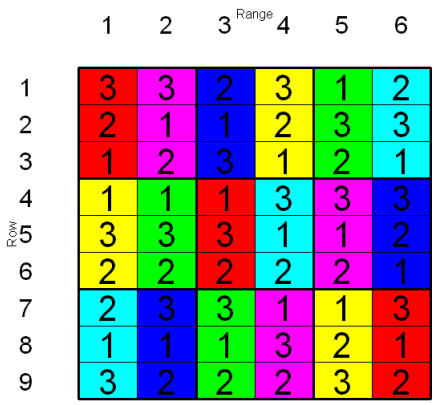


Figure 11: Split-plot design with spatially optimised main plots (coloured) and restricted optimised interaction treatments (subplot treatments labelled).

Consider a trial with 4 row spacing treatments which must run across all columns in the design. The full design is [16×6] with replicates [16×2]. The 8 other treatments whose randomisation is unrestricted are combinations of 4 varieties by 2 densities. A strip treatment will be applied to one row across the design. The following code creates treatment names with other treatments nested within row spacings and calls `des.strip00`.

```
> vnam <- c("Variety1","Variety2","Variety3","Variety4")
> rsnam <- c("20cm","30cm","40cm","50cm")
> dnam <- c("D25","D40")
> tnam <- levels(interaction(expand.grid(dnam,vnam,rsnam),sep="_"))
> ddvr <- des.strip00(nstrip.trt=4,
+                     nother.trt=8,
+                     nrd=16,ncd=6,nrr=16,ncr=2,
+                     rows.in.block=4,
+                     TreatmentName=tnam
+                     )
...
> View(ddvr$dlist)
> write.csv(ddvr$dlist,file="RSpaceStrip.csv",
+           row.names=FALSE,quote=FALSE)
```

A plot of the design is automatically produced by the function (Figure 12) with strip treatments colour coded and other treatments numbered. Note that each set of 4 rows contains a set of the strip-plot treatments.

5.5 `DiGGer` for p-rep designs.

A p-rep design is a design with p% of the treatments of interest replicated (Cullis et al., 2006, Smith et al., 2006). Other treatments of interest may be unreplicated because

- there is a not enough experimental material for replication
- it is too expensive to process replicates of all treatments

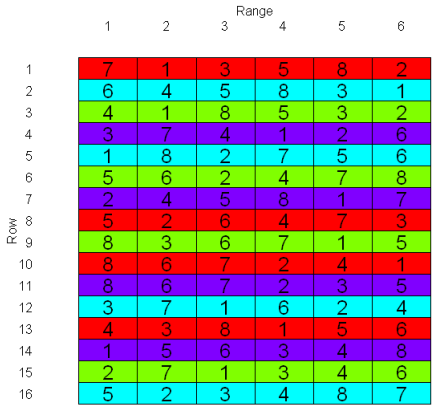


Figure 12: Strip-plot design with spatially optimised strip plots (coloured) and restricted optimised interaction treatments (other treatments labelled).

The aim of these experiments is to select the most promising entries from the test treatments for further testing in replicated trials.

Unreplicated designs which have none of the treatments of interest replicated may be considered a limiting case of a p-rep design (p=0%). In these designs check treatments provide the replication which allows data from the trial to be analysed. The treatments of interest are predicted with the poorest precision. The minimum percentage of experimental units occupied by replicated treatments should be 20%.

The aim of p-rep designs is to replace replicates of check treatments with replicates of the treatments of interest allowing the experiment to be analysed while improving the predictions of treatments of interest.

Replication of treatments across a number of sites should be balanced to equalise the precision of treatment predictions. Consider a trial where 360 entries are to be grown at 3 sites. A p-rep replication scheme that provides the minimum replication at each site and balances information across sites is given in Table 1.

Treatment	Site 1	Site 2	Site 3	Total rep
1:40	2	1	1	4
41:80	1	2	1	4
81:120	1	1	2	4
121:360	1	1	1	3
Plots	400	400	400	

Table 1: Replication levels of 360 treatments across 3 sites.

`DiGGer` may be used to optimise the designs a site at a time. The aim here is to ensure that each site is able to be analysed and that the predictions of test treatments at a site are efficient.

5.5.1 Optimisation at a single site

Design optimisation where some treatments are unreplicated present problems because

- the information about treatments is unbalanced. The efficiency of the design depends on the comparisons being made. The average efficiency of all comparisons `A++` is only appropriate

if all the treatments are test lines. When replicated check treatments are present it is appropriate to code test treatments only with a common group code and optimise comparisons within that group. If test lines are coded with group 1 `AType=A11` should be used.

- the optimisation problem is larger than necessary because swaps between unreplicated lines do not change the efficiency of the design but increase the number of possible swaps which may be tested.

Block designs with additional singly replicated treatments are augmented designs. The singly replicated treatments are referred to as secondary treatments.

Herzberg and Jarrett (2007) show that the A-optimal design for augmented designs with equal numbers of secondary treatments in blocks is the A-optimal block design for the replicated treatments. A sensible strategy for optimising p-rep designs would be to optimise a block design for replicated treatments and augment this design with secondary treatments. Blocks in the design should have near equal numbers of secondary treatments.

The design may be split into blocks using `DiGGer`'s blocking sequence mechanism. Three typical sets of blocks may be

- half trials with one unit of each duplicated entry
- check blocks with at most one unit of the highest replicated entry
- target blocks with near equal numbers of replicated entries representing the smallest section of the design which must contain replicated entries.

The first set of blocks produce replicate blocks of the replicated treatments. The second set of blocks are incomplete blocks which reduce the possibility of self-diagonals in the spatially optimised design. The target blocks ensure that replicated treatments are present in small sections of the design, often part of a column is a sensible choice.

Optimisation is slow when `DiGGer` is used directly on p-rep designs. The distribution of replicated entries to blocks is handled more efficiently by

- optimising the reduced block design for replicated treatments
- augmenting the block design with the singly-replicated treatments
- spatially optimising the augmented design
  - for comparisons between test treatments
  - only swap treatments within the final target blocks

A draft function for p-rep designs has the form:

```
des.prep00 <-
  function(nt,            # NumberOfTreatments
           nrd,           # RowsInDesign
           ncd,           # ColumnsInDesign
           trep,          # vector length nt of trt replications
           tgrp,          # vector length nt of trt groups !1=tests
           tnam=NULL,     # vector length nt of trt names
           tnum=NULL,     # vector length nt of trt number id's
           ribs,          # vector of row specs for block sequence
           cibs,          # vector of col specs for block sequence
           maxint=20000)  # max interchanges in final search
```

The vectors `trep` and `tgrp` give the `nt` treatment replications and group codes for the trial. Test treatments, the treatments of interest, in the design must be coded with group code 1. The trial must have some unreplicated treatments. The function when called

- runs the search to obtain an optimised block design for replicated treatments
- augments the block design with unreplicated treatments
- sets up a search using `A11` to spatially optimise the augmented design. Swaps are restricted to within augmented blocks.

The `run` method must be used to run the final search.

Group 1 treatments of interest

- may define all treatments in the design
- may have replicated and unreplicated treatments
- may have only unreplicated treatments

The arguments `ribs` and `cibs` contain vectors of row and column dimensions equating to `RowsInBlockSequence` and `ColumnsInBlockSequence` in the standard call to `DiGGer`. Elements of these vectors are paired to provide the nested blocking sequence which determines the way the block design for replicated treatments is blocked.

An example with 181 treatments arranged [8×27] has treatment information given in Table 2

Treatment	Replication	Group
1:150	1	1
151:175	2	1
176:177	2	2
178:181	3	2
Plots	216	

Table 2: Replication and Group in a p-rep design.

There will be 66 of 216 experimental units with replicated treatments. The trial may be conveniently split into halves, [4×27] which could have one of each duplicated treatment. The half trials could each be split into [4×9] blocks. Each of these blocks should contain 11 of the 66 replicated treatments. Making target blocks part of a single column suggests blocks here of [4×1].

This blocking sequence would be specified in a call to `des.prep00` with `ribs=c(4,4,4)` and `cibs=c(27,9,1)`. The block sequence specifications and the effective blocks for the full design are

- Phase 1: `c(4,27)`, [4×27]
- Phase 2: `c(4,9)`, [8×9], [4×9]
- Phase 3: `c(4,1)`, [8×1], [4×1]

The reduced design for replicated entries is calculated in the function to be [4×27] with a blocking sequence `c(2,27)`, `c(2,9)` and `c(2,1)` corresponding to the blocking sequence in the main design. Note that this reduced design pads the 66 replicated plots with 42 missing plots. The [2×1] target blocks in this reduced design are augmented with 2 additional missing plots and all the missing plots are replaced with a random allocation of the unreplicated treatments.

The call to the function is:

```
> dp181 <- des.prep00(nt=181, nrd=8, ncd=27,
+                     trep=rep(c(1,2,2,3),c(150,25,2,4)),
+                     tgrp=rep(c(1,1,2,2),c(150,25,2,4)),
+                     ribs=c(4,4,4),
+                     cibs=c(27,9,1))
```

```
...
> dp181 <- run(dp181)
...
> mp181 <- getDesign(dp181)
> des.plot(mp181,seq(150),col=8,new=TRUE,label=FALSE)
> des.plot(mp181,seq(25)+150,col=7,new=FALSE,label=TRUE,chtdiv=3)
> des.plot(mp181,seq(2)+175,col=4,new=FALSE,label=TRUE,chtdiv=3)
> des.plot(mp181,seq(4)+177,col=5,new=FALSE,label=TRUE,chtdiv=3,
+          bdef=cbind(4,9),bwd=4)
> des.tab(mp181,4,27)

  1   2
104   2

  1   2
104   2
```

The treatment frequency table shows the design has been split into "replicate" blocks. The optimised design for this example is given in Figure 13.

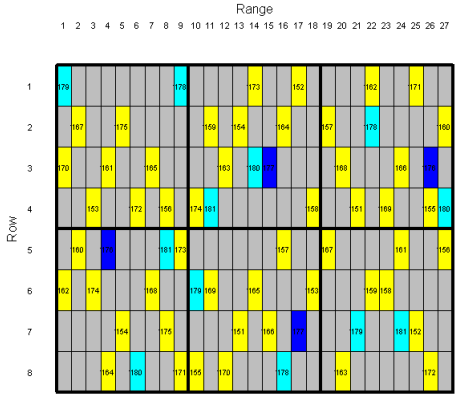


Figure 13: Optimised position of replicated test (151:175) and check (176:181) treatments in a p-rep design.

5.5.2 Additional controls in a replicated design revisited

Returning to the design described in Section 5.3, the placement of control treatments can be made with a call to `des.prep00`. Recall that there are 15 treatments, with the first 14 occurring once in each replicate and treatment 15 occurring 4 times in each replicate. Placement of the control treatment may be considered a sampling question: where can 16 control treatments be placed to give most information about other plots in an [18×4] array of plots?

The first step is to treat the problem as an unreplicated design with 1 treatment replicated 16 times and 56 unreplicated treatments in a design with a major blocking structure defined by [9×2] corner blocks. Calling `des.prep00` produces a design for the control treatment.

```
> du57 <- des.prep00(nt=57, nrd=18, ncd=4,
+                    trep=rep(c(16,1),c(1,56)),
```

```
+                    tgrp=rep(c(2,1),c(1,56)),
+                    ribs=c(9,9),cibs=c(2,1))
> du57 <- run(du57)
```

The control treatment can then be kept fixed and the other treatments added by replicate.

```
> mu57 <- getDesign(du57)
> s15 <- findblks(18,4,9,2)
> s15[mu57==1] <- 15
> m15 <- s15
> m15 <- s15
> for (i in seq(4))
+     m15[s15==i] <- sample(14)
+
> d15f <- DiGGer(15, 18,4, 9,2,
+                TreatmentRep=rep(c(1,4),c(14,1)),
+                InitialDesign=m15,
+                InitialSwap=s15)
> d15f <- run(d15f)
```

Figure 10 shows the optimised design with spaced check locations produced by this approach.

5.6 Embedded p-rep designs

Consider a standard replicated variety trial. Quality testing is to be carried out on a selection of plots from the trial. Costs are to be kept down by limiting the number of plots tested. The selection must include a complete replicate of the entries and must also include a proportion of duplicate entries so that field effects on quality can be modelled. The modelling of field effects on quality will be enhanced if the selection is a set of contiguous plots.

An embedded p-rep design (Cullis submitted) is a set of contiguous plots in a rectangular array containing a replicate of entries and a proportion of duplicated entries.

A function `des.embedopr00` uses two additional arguments,
`RowsInEmbeddedBlock` and `ColumnsInEmbeddedBlock`,
to specify the section of a design which is to contain the embedded p-rep design.

```
des.embedopr00 <- function(NumberOfTreatments,
                     RowsInDesign,
                     ColumnsInDesign,
                     RowsInReplicate,
                     ColumnsInReplicate,
                     B2D=TRUE,
                     RowsInEmbeddedBlock,
                     ColumnsInEmbeddedBlock,
                     InitialDesign=NULL,
                     ...)
```

The function takes the specifications for a standard replicated `DiGGer` search and specifies an additional block placed at `(Row 1, Column 1)` which will be optimised during the search. If the block contains more units than a replication then the optimal blocking should place a replicate and a few duplicate treatments in the block.

Consider a trial with 32 entries arranged [8×12] with replicates [8×4]. A set of contiguous plots [4×9] is to be used as an embedded p-rep design. The call to the function is

```
d32 <- des.embedopr00(NumberOfTreatments=32,
                      RowsInDesign=8,
                      ColumnsInDesign=12,
```

```
                RowsInReplicate=8,
                ColumnsInReplicate=4,
                RowsInEmbeddedBlock=4,
                ColumnsInEmbeddedBlock=9)
```

`des.embedopr00` makes several calls to `DiGGer` to optimise the embedded design:

- optimize treatments in the embedded block
- optimize the excised p-rep design subject to replication constraints
- fix the position of treatments in the p-rep design while optimising the remaining treatments subject to replication constraints.

The call to the function produces a plot of the design (Figure 14).

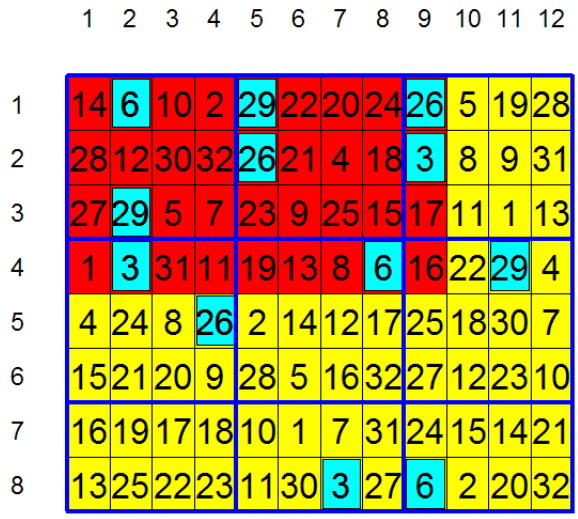


Figure 14: Embedded p-rep design in red with duplicate treatments shown in cyan.

Note that this is a limited implementation of the embedded p-rep design concept. It is restricted to blocks anchored at Row 1, Column 1. The overlap of the embedded block with the replicates needs to be carefully planned because of the implications on the location of duplicate entries within an embedded block. For example, choosing the embedded block to be $[\ 8 \times 5\ ]$ in the example above necessarily results in all units of column 5 having duplicate entries. This is not the optimal arrangement for a p-rep design with these dimensions.

References

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.

A.B. Smith, P. Lim, and B.R. Cullis. The design and analysis of multi-phase plant breeding trials. *Journal of Agricultural Science*, 114:393–409, 2006.