Designs on the Web

R. A. Bailey [a] Peter J. Cameron [a,1] Peter Dobcsányi [a]
John P. Morgan [b,2] Leonard H. Soicher [a]

[a] *School of Mathematical Sciences, Queen Mary, University of London, London E1 4NS, U.K.*

[b] *Department of Statistics, Virginia Tech, Blacksburg, VA 24061-0439, USA*

Abstract

In 2001, the United Kingdom Engineering and Physical Sciences Research Council awarded three of the authors of this paper a grant for "A Web-based resource for design theory". As the project developed, we have had to face a number of problems, ranging from fundamental questions such as "What is a design?", through research topics such as "How should the concept of partial balance be extended to designs which do not have constant block size?", to more practical problems concerning what format we should use to store designs. This paper gives a brief description of the project to date, concentrating on theoretical questions about designs and their classification.

Key words: design, block design, XML, partial balance, robustness, regular-graph design

1 The DTRS project

The authors are in the process of developing a web-based Design Theory Resource Server (DTRS) for combinatorial and statistical design theory. The project is funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant GR/R29659. The project has a number of aspects:

- the development of a database of designs;
- software for constructing and manipulating designs, and investigating their automorphism groups, resolutions, optimality properties, etc.;

[1] Corresponding author

[2] Supported by NSF grant DMS01-04195

- documentation, partly in the form of an *Encyclopaedia of Design Theory* [8] which will explain the types of design, their uses, and their representations in the database.

One critical element is the *External Representation of Designs* which will be used to store designs and their properties in a standard platform-independent manner ('external' means external to any software). This will allow for the straightforward exchange of designs between various computer systems, including databases and web servers, and combinatorial, group theoretical and statistical packages. The external representation will also be used for outside submissions to our design database.

The first official release of the *External Representation* with documentation has appeared [9]. This will be described below.

2 What is a design?

A typical situation facing a statistician is as follows. We have a set T of treatments which we wish to compare, and a set Ω of plots or experimental units to which the treatments may be applied. A *design* is a function $F : \Omega \to T$ (so that $F(\omega)$ is the treatment applied to plot ω).

If T and Ω are completely unstructured, one should simply use each treatment the same number of times (as near as possible). However, the existence of structure on the sets T and Ω complicates matters!

For example, one treatment may be a placebo, or the "standard" treatment against which the others are to be compared; or the treatments may have a number of factors which can be applied at different levels (fertiliser dose, watering, planting time, etc.) These are important, and must be considered eventually; but in order to start somewhere, we decided to disregard them at first. If T is homogeneous, we can replace the function F by a partition of Ω (and assign one treatment to each part).

Plot structure arises because the set of plots is not homogeneous. For example, we may have several plots on each of a number of farms in different geographical areas; soil fertility or moisture content may vary across a field; and experiments over a period of time are subject to daily or seasonal effects. Plot structure may include the potential for treatments to interfere with each other if they are neighbours: for example, one plant may shade another, and drugs may have carry-over effects.

The simplest plot structure is a system of blocks, where plots in a block are

more alike than those in different blocks (as in the above example of farms). Thus, the blocks form a partition of Ω. So a *block design* consists of a set Ω with two partitions, corresponding to treatments and blocks.

The block partition is given, and we have to choose the treatment partition so that information can be recovered about the treatments as efficiently as possible – this means that the variances of the estimators of treatment differences should be as small as possible.

In the worst case, if we applied each treatment on only one farm, we couldn't separate treatment effects from geographical effects: these effects would be *confounded*.

More generally, there is nothing to stop us from using the same treatment more than once in a block. Indeed, this may be advantageous in some cases. For example, in clinical trials, where a block consists of a single patient for a number of time periods, treatments may have carry-over effects into the next time period. However, if our aim is to recommend a single "best" treatment, this will then be used all the time, and we have to know about how it interacts with its own carry-over effect!

A design is called *binary* if each treatment occurs at most once in each block. Binary designs are simpler to study than general designs, and most of the mathematical results apply only to such designs. We set up the notation so that we can handle general block designs, but for the most part we consider only binary designs; if not stated otherwise, assume that all designs are binary.

If the number of treatments is equal to the number of plots in a block, we could ensure that each treatment occurs once in each block. This is a *complete* block design, and is clearly the best we can do. Often, the constraints of the experiment don't allow this.

Typically, the block size k is smaller than the number v of treatments, so that the block design is *incomplete*. In this case, if there is a (binary) block design which is *balanced* (so that any two treatments occur together in a block exactly λ times), then it is the best design to use, with respect to all practical criteria. This is why balanced incomplete-block designs (BIBDs, or 2-designs) are so important. But often it is the case that no such design exists, and indeed there may be no design which is uniformly best. In this case the choice of design might depend on other factors.

Block designs already form an interesting and extensive category. Even within the class of binary designs with constant block size and constant replication number, we find much of interest to the combinatorialist: balanced designs include projective and affine planes, unitals, Steiner systems, and difference sets, while non-balanced designs include partial geometries, generalized polygons, near polygons, group-divisible and partially balanced designs, and so on.

Combinatorialists usually refer to treatments as "points", as we will mostly do from now on.

Moreover, many other kinds of design can be regarded as block designs. A Latin square is often treated as a *square lattice design*, whose points are the cells of the square, and whose blocks correspond to the rows, columns, and symbols. This immediately extends to families of mutually orthogonal Latin squares, and to semi-Latin squares.

Our interpretation of block designs lends itself to several natural generalisations. If the blocks of a block design comprise a multiset of multisets of points, we may consider designs consisting of several multisets of multisets of points. These include affine and projective geometries (where the blocks are subspaces of various dimensions), and configurations in algebraic geometry (where the blocks are algebraic varieties of various kinds). Alternatively, we could take a multiset of multisets of multisets; this leads to iterated block designs such as *nested block designs* (with small blocks nested in large ones) and whist tournaments (with rounds consisting of tables consisting of players). Resolved designs could be considered under this heading.

Regarding a block design as a bipartite incidence graph, we replace bipartite graphs by multipartite graphs. *Buekenhout geometries* [6] form an important class here.

Regarding a block design as a set of plots with a block partition and a treatment partition, the natural generalisation is to a set carrying a family of partitions. These occur as *chamber systems* in the work of Tits [27], and also include a number of types of statistical design such as orthogonal arrays, orthogonal main effects plans, semi-Latin squares, Youden squares, double Youden rectangles, Freeman–Youden rectangles, SOMAs, Howell designs and Room squares.

Still greater generality arises from the possibility of imposing structure on one or more of the multisets. For example, the plots in each block may be structured, as in the Oberwolfach problem and one version of whist tournaments. The set of blocks, or of treatments, or of plots, may be structured: partially

balanced designs and row-column designs are of this type.

A detailed account of the proposed classification is in preparation [2].

3 Representations of block designs

3.1 Partitions, graphs, block lists

In what follows, a *multiset*, or *list*, informally denotes a set of elements with positive integer multiplicities. We write multisets in square brackets with elements repeated the appropriate number of times. The order of the elements is not significant.

A block design can be represented in several different ways:

- $\mathbf{R}_1$: A set of plots with two partitions, a block partition and a treatment partition (as described above).
- $\mathbf{R}_2$: A bipartite graph, whose vertices are treatments (also called *points*) and blocks. An edge joins two vertices if one is a treatment and the other a block such that the treatment appears in the block; the edge is labelled with the number of times the treatment appears in the block. This is the *incidence graph*, or *Levi graph*, of the design. Note that, if the design is binary, then all the edge labels are equal to 1 and may be omitted.
- $\mathbf{R}_3$: A set of points, with each block represented as a multiset of points (recording the occurrences of the treatments in that block). Thus the blocks form a multiset of multisets.
- $\mathbf{R}_3'$: Dual to $\mathbf{R}_3$.
- $\mathbf{R}_4$: A bipartite graph with vertices and edges labelled. The vertices are equivalence classes of points and blocks, two blocks being equivalent if they contain the same points with the same multiplicities, and dually. The edge labels are as above. The vertex labels denote the cardinalities of the corresponding equivalence classes.

Example 1 Suppose we have six plots, $1, 2, 3, 4, 5, 6$, which fall into two blocks $\alpha = \{1, 2, 3, 4\}$ and $\beta = \{5, 6\}$. We have two treatments A and B, and apply A to plots $1, 3, 5$ and B to plots $2, 4, 6$. These two partitions form the representation $\mathbf{R}_1$ of the design. It has 8 automorphisms.

The representation $\mathbf{R}_2$ consists of a complete bipartite graph on vertices $\{A, B\}$ and $\{\alpha, \beta\}$, with multiplicities 2 on $A\alpha$ and $B\alpha$ and 1 on the other two edges. This has 2 automorphisms.

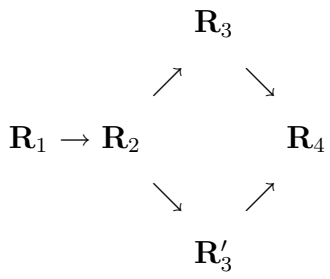
The representation $\mathbf{R}_3$ has point set $\{A, B\}$, blocks $[A, A, B, B]$ and $[A, B]$.

Again there are 2 automorphisms.

On the other hand, $\mathbf{R}_3'$ has point set $\{\alpha, \beta\}$ and a single block $[\alpha, \alpha, \beta]$ with multiplicity 2. It has trivial automorphism group.

Finally, $\mathbf{R}_4$ is the complete bipartite graph $K_{1,2}$ with vertex A labelled 2, and α and β labelled 1; edges $A\alpha$ labelled 2, $A\beta$ labelled 1. Its automorphism group is also trivial.

We have the following schematic relation between the representations:



The arrows correspond to homomorphisms between the automorphism groups. The first homomorphism measures the non-binarity of the design; its kernel consists of the permutations of the set of plots which fix part-wise the meet of the treatment and block partitions. The other homomorphisms measure repeated blocks or points in a similar way.

For the DTRS project, at present we consider only binary designs. We have chosen to use representation $\mathbf{R}_3$, which is almost universal among mathematicians. Thus, a block design for us consists of a set of points, and a list of blocks, each block a subset of the point set; but repeated blocks are allowed.

3.2 Matrices

Let n be the number of plots, b the number of blocks and v the number of treatments. The incidence of plots in blocks can be recorded by an $n \times b$ matrix X_B whose (ω, m) entry is equal to 1 if plot ω is in block m, and equal to zero otherwise. The allocation of plots to treatments is shown in an analogous $n \times v$ matrix X_T, with (ω, i) entry 1 if $F(\omega) = i$ and 0 otherwise, where F is the design function.

Now the matrix $X_T^\top X_B$ is the $v \times b$ matrix whose (x, m) entry is equal to the number of plots in block m which receive treatment x. In the case of a binary design, this is a zero-one matrix, and is the familiar incidence matrix of the design.

The $v \times v$ matrix $X_T^\top X_B X_B^\top X_T$ is called the *concurrence matrix* Λ of the design: its (i,j)-entry λ_{ij} is equal to the number of ordered pairs of plots (α, ω) for which (i) α and ω are in the same block, (ii) $F(\alpha) = i$, and (iii) $F(\omega) = j$. If the design is binary then λ_{ii} is equal to the *replication* of treatment i: that is, the number of plots which are allocated treatment i. Also, if the design is binary then, for $i \neq j$, the concurrence λ_{ij} is equal to the number of blocks in which treatments i and j both appear. So, in the familiar case of a balanced incomplete-block design, $\Lambda = (r - \lambda)I + \lambda J$, where J is the all-1 matrix.

Let Q be the $n \times n$ matrix which gives orthogonal projection onto the orthogonal complement of the column space of X_B. Thus $QX_B = O$, $Q^\top = Q$ and $Q^2 = Q$. It can be shown that

$$Q = I - X_B K^{-1} X_B^\top,$$

where K is the diagonal matrix whose entries are the block sizes.

The *information matrix* C of the design is defined by

$$C = X_T^\top Q X_T.$$

Note that the row sums of C are all zero, so that the all-1 vector is in the kernel of C. Thus C is not invertible. In order to determine estimator variances for the experiment (in loose statistical parlance, "information" is the inverse of variance), the statistician uses the *Moore–Penrose inverse* C^+ of C, where, if C has spectral decomposition

$$C = \sum_\lambda \lambda E_\lambda,$$

where λ runs over the eigenvalues of C, and E_λ is the orthogonal projection onto the λ-eigenspace, then

$$C^+ = \sum_{\lambda \neq 0} \lambda^{-1} E_\lambda,$$

so that $CC^+ = C^+C$ is the orthogonal projection onto the image of C.

The information matrix can often be simplified. If all blocks have size k then

$$Q = I - \frac{1}{k} X_B X_B^\top$$

and so

$$C = X_T^\top X_T - \frac{1}{k}\Lambda.$$

The matrix $R = X_T^\top X_T$ is diagonal: if the design is binary then its entries are the replications of the treatments. In particular, if every treatment has replication r then

$$C = rI_v - \frac{1}{k}\Lambda.$$

The matrix $R^{-1/2}CR^{-1/2}$ is also important [7, Section 8.2]. In the equireplicate, constant block size, case, it is simply $I_v - (1/rk)\Lambda$.

4 The External Representation of designs

The concept of External Representation of designs can be best understood through its role in the operation of the Design Theory Resource Server (DTRS). DTRS has many faces; it will be an ever-growing database of designs, application server, web server for design related online documents, software repository, etc. The main purpose of the external representation is to provide a platform-independent method for information exchange about designs. In other words, the external representation acts as a *communication protocol* specialized for "talking about designs". This protocol is used in communication between various components of DTRS and its users. Here the concept "users" covers both human and software agents. Some examples for such communicating agents are database back-end for storing designs, middle layers between the database and the web and/or application servers, a researcher uploading some particular collection of designs, a user searching for designs having given properties, and a statistical application program directly accessing the DTRS database. While these agents are free to use any internal representation of designs, they must use the standard external representation when they communicate with each other.

The external representation is used in three main areas:

(1) An external representation is a formalism to encode various classes of designs as mathematical objects together with their most important properties. Many of these properties are complex mathematical objects in their own right.
(2) The external representation can define invariants of a given list of designs. The use of such invariants provides a method for formulating complex

queries about designs. A query will be expressed in terms of list invariants and the reply to this query will be a list of designs satisfying these invariants.

(3) The external representation will be used as a specification tool to determine the content and, to some extent, the structure of the DTRS design database. Note, however, the database's internal representation can (and probably will) be quite different from this format.

Based on the above functionalities, we have determined the main technical requirements for an external representation as follows.

- It can express the particular mathematical structures.
- It represents a hierarchical structure: a rooted, labelled tree.
- It is hardware/software platform independent and text based.
- It can be easily parsed.

Satisfying the requirements outlined above, one can think of many different implementations fitting the bill. We mention here two possibilities: the oldest, Lisp S-expressions; and the recently most popular, XML. Out of practical considerations, we have decided to use XML. XML documents are human-readable to a limited extent, but are primarily intended for computer processing.

The External Representation document contains a specification of the syntax together with extensive documentation of the mathematical semantics of the represented designs.

When the design database is set up, the format of designs in the database will not necessarily be the same as the specification in the External Representation; however, an XML file is searchable, so a simple database can be implemented using this format.

Example 2 This is a list of designs containing a single design, the Fano plane, in XML, in our external representation format.

```
<list_of_designs
 design_type="block_design" dtrs_protocol="1.0"
 no_designs="1" pairwise_nonisomorphic="true"
 xmlns="http://designtheory.org/xml-namespace">
    <block_design b="7" id="t2-v7-k3-L1-1" v="7">
        <blocks ordered="true">
            <block><z>0</z><z>1</z><z>2</z></block>
            <block><z>0</z><z>3</z><z>4</z></block>
            <block><z>0</z><z>5</z><z>6</z></block>
            <block><z>1</z><z>3</z><z>5</z></block>
            <block><z>1</z><z>4</z><z>6</z></block>
            <block><z>2</z><z>3</z><z>6</z></block>
            <block><z>2</z><z>4</z><z>5</z></block>
```

```
        </blocks>
    </block_design>
</list_of_designs>
```

The outermost tag is `list_of_designs`. Here, in due course, we shall be able to store such information as "these are, up to isomorphism, all designs with such-and-such properties" (or maybe "the designs in the list are pairwise non-isomorphic but we don't know if the list contains all designs with the specified properties"), or "these designs were constructed by X using software Y". Other comments can also be added. Each design in the list is described by the earlier specification.

Any document which is exchanged by systems following this specification should be a `list_of_designs`.

At the first level of indentation, between the opening and closing tags `block_design`, we have indeed specified the design: it has $v = 7$ points, $b = 7$ blocks, and the seven blocks are listed (the first one is $[0, 1, 2]$, and there are tags to identify each of 0, 1, 2 as integers). There is also an identification string for the design.

The specification of a block design is shown below. Properties followed by a question mark are optional. Remember that all designs are assumed to be *binary*, so that a block is a set of points (rather than a multiset). This is an excerpt from a Relax NG [24] file, a compact and friendly way to specify the structure of an XML document. The previous example included only the required fields and the attribute `b`.

```
block_design = element block_design {
    attribute id { xsd:ID } ,
    attribute v  { xsd:positiveInteger } ,
    attribute b  { xsd:positiveInteger } ? ,
    attribute precision {xsd:positiveInteger } ? ,
    blocks ,
    point_labels ? ,
    indicators ? ,
    combinatorial_properties ? ,
    block_design_automorphism_group ? ,
    resolutions ? ,
    statistical_properties ? ,
    alternative_representations ? ,
    info ?
}
```

The attributes `id`, `v` and `b` and the list of blocks have been described above. The attribute `precision` refers to storage of real numbers in the `statistical_properties` section. We now discuss the other properties listed, and also partial balance properties which we will implement in a later release.

4.1 Point labels

As explained, the points of a block design are called $0, 1, \ldots, v-1$. But many block designs are constructed from finite geometries, graphs, or other block designs, and the points can be given labels reflecting their origin in such a construction. For example, if the point set is a subset of a finite vector space, the point labels may be the coordinates of the corresponding vectors.

4.2 Indicators

Indicators give quick information about the design, usually expanded in other sections. They include `repeated_blocks`, `resolvable`, `affine_resolvable`, `equireplicate`, `constant_blocksize`, `t_design`, `connected`, `pairwise_balanced`, `variance_balanced`, `efficiency_balanced`, `cyclic`, `one_rotational`.

They take the value "true" or "false". There may be also some extra information. For example, the `t_design` indicator gives the maximum value of t; the `equireplicate` indicator gives the replication number. We have decided that the extra information, if any, will be a single number.

The balance conditions are defined as follows. A design is *pairwise balanced* if any two points lie in the same number of blocks; that is, the off-diagonal elements of the concurrence matrix are all equal. It is *variance balanced*, resp. *efficiency balanced*, if the information matrix C (resp. $R^{-1/2}CR^{-1/2}$) has eigenvalue 0 with multiplicity 1 (that is, the design is connected) and the other eigenvalues are all equal. In the case of variance balance, since the all-1 vector is a null vector of C, we see that the design is balanced in this sense if and only if C is *completely symmetric* (that is, the diagonal elements are constant and the off-diagonal elements are also constant). See [25] for further details.

In the case of efficiency balance, it can be shown that the common eigenvalue of $R^{-1/2}CR^{-1/2}$ is given by

$$\epsilon = \frac{n(n-b)}{n^2 - \mathrm{Trace}(R^2)}, \tag{1}$$

where $n = \mathrm{Trace}(R)$ is the number of plots, and that the equation

$$R^{-1}CR^{-1} - \epsilon R^{-1} = -\epsilon J$$

holds.

4.3 Combinatorial properties

A t-(v, k, λ) design, or t-design for short, is a (binary) block design with v points, in which every block contains exactly k points, and any t points are contained in exactly λ blocks, where $\lambda > 0$. Properties in this section are mainly relevant to t-designs with $t \geq 2$: these are also the (binary, constant block size) *balanced incomplete-block designs*, or BIBDs.

The properties recorded include a range likely to be of interest to mathematicians. These include Steiner system, square design, projective or affine plane. (We have chosen to use the term "square design" for one with equally many points and blocks, rather than "symmetric design", first because no symmetry of the incidence matrix is implied, and second because the term "symmetric design" sometimes implicitly requires the design to be balanced and have constant block size.) We also include information about α-resolvability, t-wise balance, etc.

We refer to [4] for further details.

4.4 Automorphism group

As noted earlier, the precise definition of an automorphism of a block design depends on the method of representation. In our representation where the blocks form a list of sets of points, an *automorphism* of the design is a permutation of the points which maps each block to a block with the same multiplicity in the list. The automorphisms of a design form a group, the *automorphism group* of the design.

Since the automorphism group is a permutation group, we have given in the document a specification for permutation groups. By convention, a permutation group acts on the set $\{0, 1, \ldots, n-1\}$ for some n; the elements may be indices for the points of a design, and we define the domain of the permutation group to be `"points"` accordingly. (This can easily be expanded to cope with actions on other sets of interest, such as blocks, or flags, or pairs of points, as required.) We store the degree, the order, the domain, and a list of generators for the permutation group as compulsory properties. Optional properties include the number of orbits, permutation rank, primitivity and multiple transitivity, as well as certain properties related to partial balance (generously transitive, multiplicity-free and stratifiable). We may also give cycle types of elements together with a representative element for each cycle type.

The generators can be exported to a program such as GAP [13], where further properties of the automorphism group can be investigated. Of course, unlike some other properties, there is no unique choice of generators for a permutation group.

4.5 Resolutions

A *resolution* or *parallelism* of a block design is a partition of the blocks into *resolution classes* or *parallel classes*, such that each parallel class is a partition of the point set.

If a design is resolvable, we can in principle store representatives of all isomorphism classes of resolutions (two resolutions being isomorphic if one is obtained from the other by applying an automorphism of the design).

4.6 Statistical properties

Among the statistical properties, we include canonical and pairwise variances, how the design performs on a number of optimality criteria, the efficiency factors of the design, and robustness properties. We describe these in turn.

The information matrix C has an eigenvalue 0; the design is connected if and only if the multiplicity of 0 as an eigenvalue is precisely 1. (If a design is not connected, then contrasts between treatments in different components cannot be estimated.) The $v-1$ non-zero eigenvalues of the Moore–Penrose inverse C^+ of C are the *canonical variances* of the design (if the design is disconnected, we regard some of the $v-1$ canonical variances as being infinite). The canonical variances are the variances of the estimators of a set of $v-1$ normalized treatment contrasts, which are linear functions comparing treatment efficacy, with coefficients given by the unit eigenvectors of C.

In some experiments, rather than general contrasts, interest lies solely in the collection of $v(v-1)/2$ pairwise differences of treatments. The *pairwise variances* are the variances of estimators of treatment differences.

One (key) goal of statistical design is to make variances of estimators small, and one way in which a design is said to be optimal is to achieve smallest variance in some sense. Another common (typically supporting) goal is to make variances have comparable magnitude, that is, to balance the variances as far as possible. Many of the popular design optimality criteria, from both the variance size and variance balance points of view, are based on the canonical variances, so we record these and a number of functions of them. Among others we include their arithmetic mean (the A-criterion), their maximum (the E-criterion), their product (the D-criterion), the number of distinct values, and the ratio of largest to smallest. Comparable criteria based on the pairwise variances, when distinct from those based on canonical variances, are also employed, for instance their maximum (the MV-criterion), the number distinct, and so on. When for a given criterion a design is known to be best in the universe of all binary designs with the same v, b, and block sizes, it is recorded as having *absolute efficiency* of 1. The absolute efficiencies of all other recorded designs in this universe are reported relative to the optimal value, this

necessarily being a number no greater than 1 for any binary design.

The *canonical efficiency factors* are the eigenvalues of $R^{-1/2}CR^{-1/2}$, where R is the diagonal matrix whose entries are the treatment replication numbers (if the replication number r is constant, these are just r^{-1} times the inverses of the canonical variances). Efficiency factors, which are between 0 and 1 inclusive, measure the quantity of statistical information in a design relative to what can be achieved with the same replications in an unblocked design. Certain functions of the efficiency factors are used, analogously to the above optimality criteria, for choosing a good design in the class with the prescribed replication numbers. Examples include the geometric and harmonic means, and the minimum.

Statistical robustness of a block design is defined as its ability to maintain desirable statistical properties under loss of individual plots or entire blocks. A catastrophic result of such a loss is that the design become disconnected. Less than catastrophic but of genuine concern are losses in the information provided by the design, as measured by various optimality criteria. We record such information as "The design is connected under all possible ways in which m plots (or m entire blocks) can be lost", or "If m plots (or m entire blocks) are lost, the value of an optimality function of the design is so much"; the latter statement is made for both the worst case over all such losses and the average over all such losses, and the functions employed are now called *robustness criteria*. For each robustness criterion value, we record the absolute efficiency relative to the best value (when known) over all "reduced" designs with the same v, b, and block sizes. We also record the *self efficiency*: the robustness value relative to the criterion value for the same design with no loss of observations.

Typically, measures of variance balance, along with properties like constant replication or block size and more generally t-design properties, are not robust, and so these are not used as robustness criteria.

4.7 Partial balance properties

In the case of partial balance properties, everybody agrees what it means for a binary design with constant block size to be partially balanced, but beyond that there are conflicting views. We have tried to clarify the situation below. This requires more work, so this is not included in the current release of the external representation; but we expect it to be added subsequently.

A *coherent configuration* C on a set Ω is a partition of Ω^2 with the properties that the diagonal is a union of parts, the transpose of a part is a part, and the span $A(C)$ of the basis matrices corresponding to the parts is closed under matrix multiplication. Now the join (in the partition lattice) of coherent configurations is a coherent configuration. (This follows from the fact that $A(C_1 \vee C_2)$ is the intersection of $A(C_1)$ and $A(C_2)$. The result is due to Higman [15]; this short proof is due to Bailey [1].)

A coherent configuration is *homogeneous* if the diagonal is a single part; it is *commutative* if the basis matrices commute; it is *symmetric* if the basis matrices are symmetric; and it is *trivial* if the complement of the diagonal is a single part. These conditions become stronger in the order stated. A symmetric coherent configuration is the same thing as an *association scheme* (as the term is commonly used; unfortunately, some authors [11,3,12] use it to mean one of the three possible generalisations.)

The homogeneous coherent configurations on up to 30 points have been listed by Hanaki and Miyamoto [14].

Note that if C is a coherent configuration, then $A(C)$ is closed under taking inverses of invertible matrices. In the case of a symmetric coherent configuration, Moore–Penrose inverses can be defined as previously described, and $A(C)$ is closed under taking Moore–Penrose inverses. This is important for the statistical applications, which as we have seen involve inverting the information matrix of a design; the algebraic apparatus of coherent configurations makes this job easier.

For any square matrix M with rows and columns indexed by Ω, the *balancer* of M is the coarsest coherent configuration C such that M is constant on the classes of C. (This exists since it is the join of all coherent configurations with this property, and there is certainly at least one such configuration, namely the partition of Ω^2 into singletons.)

Now define M to be **h**-*balanced*, **c**-*balanced*, **s**-*balanced*, or **t**-*balanced* respectively if its balancer is homogeneous, commutative, symmetric, or trivial respectively. Each of these concepts implies the next in the list, and none of these implications reverses in general; but if a symmetric matrix is **c**-balanced then it is **s**-balanced, since the symmetrisation of a commutative coherent configuration is a coherent configuration. A matrix is **t**-balanced if it has constant diagonal and constant off-diagonal; that is, it is completely symmetric.

A *Jordan configuration* on Ω is a partition C of Ω^2 into symmetric parts with the properties that the diagonal is a union of parts and $A(C)$ is closed under the Jordan product $A \circ B = (AB + BA)/2$. It is homogeneous if the diagonal is a single part. If C is a Jordan configuration, then $A(C)$ is closed under taking generalized inverses. The motivation for using Jordan configurations in statistics is similar to that for association schemes: see Malley [19].

As before, the join of Jordan schemes is a Jordan scheme; so a symmetric square matrix M has a *Jordan balancer*, the coarsest Jordan scheme for which M is constant on the parts. (The Jordan scheme obtained by symmetrising the partition of Ω^2 into singletons is finer than any partition into symmetric parts.) We say that M is **j**-*balanced* if its Jordan balancer is homogeneous.

For a symmetric matrix, **j**-balance is implied by **h**-balance, because the symmetrisation of a coherent configuration is a Jordan configuration. We do not know whether

this implication reverses.

Problem 3 Does there exist a symmetric matrix which is **j**-balanced but not **h**-balanced?

Now consider a block design D. We make no assumption about binarity, constant block size, or anything else. If **x** denotes one of the letters **j**, **h**, **c**, **s**, or **t**, then we say that D is

- *pairwise* **x**-*balanced* if its concurrence matrix Λ is **x**-balanced;
- *variance* **x**-*balanced* if its information matrix C is **x**-balanced.

In fact we do not need the letter **c** here, since all these matrices are symmetric. Moreover, the letter **t** can always be omitted from the terminology, since each type of **t**-balance is the same as balance of that type, as we have defined in Section 4.2.

In the case of efficiency balance, things are not so clear. As we saw, the matrix $R^{-1}CR^{-1} - \epsilon R^{-1}$ is completely symmetric if (and only if) the design is efficiency-balanced, where ϵ is given by Equation (1). So we could define the various types of partial efficiency balance of the design by using this matrix in place of Λ or C. However, we have no examples which are not equireplicate, and we are not sure this is the best way to proceed. More research is needed!

If the block design is binary and equireplicate and has constant block size, then all types of balance coincide, so we can drop the descriptive adjective and speak of **j**-balance, etc. For this class of designs, **s**-balance is the usual concept of partial balance originally introduced by Bose and Nair [5].

4.8 Alternative representations

At the moment we permit a block design to be represented as an incidence matrix. Other representations can be imagined, and will be added if required. For example, a design may be the set of supports of minimum-weight words in a linear code. Also, we may give a group, and a set of *base blocks* (orbit representatives), generalising the familiar difference-set representation of square cyclic designs.

5 Software

As part of the DTRS project, a GAP package named DESIGN is being developed.

GAP [13] is a free computer system for discrete algebra, with particular emphasis on computational group theory. GAP packages are user contributions providing

well-defined additional functionality to GAP. To be *accepted* a package must go through a refereeing process. The current version of DESIGN, developed by Leonard Soicher and used internally by the DTRS project, is being released publicly (see `http://designtheory.org/software/`) and will be submitted to the GAP package refereeing process.

At present the DESIGN package deals with combinatorial and group-theoretical aspects of block designs, with particular emphasis on the classification of (sub)designs and the partitioning of block designs. The main functionality of the package is described in more detail below.

The package can construct binary block designs with small v (up to about 20), satisfying a wide range of user-specified properties. These properties include: the number v of points; the possible block sizes and (optionally) the block-size distribution; (optionally) the maximum multiplicity of a block of each size; (optionally) the replication number r; for a given t, for each t-subset U of the points, the number of blocks containing U (this number may depend on U); (optionally) the possible sizes of intersections of pairs of blocks of given sizes; a subgroup G of S_v (default $G = S_v$) to specify that G-orbits are isomorphism classes; a subgroup H of G required to be a subgroup of the automorphism group of each design (default $H = \{1\}$), and whether the user wants a single design with the specified properties (if one exists), all such designs up to isomorphism (as determined by G), or a list of such designs containing at least one representative from each G-isomorphism class.

More generally, the package can construct subdesigns of a given (not necessarily binary) block design, such that the subdesigns each have the same user-specified properties. Here a *subdesign* of D means a block design with the same point set as D and whose block list is a submultiset of the blocks of D. In this case, the default G determining isomorphism is $\mathrm{Aut}(D)$. For example, given a block design D, each of whose blocks has size k, we classify the parallel classes of D by classifying (up to the action of $\mathrm{Aut}(D)$) the subdesigns of D that are 1-$(v, k, 1)$ designs.

The package can construct partitions of (the block multiset of) a given block design, such that the subdesigns whose block multisets are the parts of this partition each have the same user-specified properties. For example, given a block design D, each of whose blocks has size k, we classify the resolutions of D by classifying (up to the action of $\mathrm{Aut}(D)$) the partitions of D into 1-$(v, k, 1)$ designs.

There are functions to determine whether two binary block designs are isomorphic, and to calculate the automorphism group of a binary block design. These functions make use of Brendan McKay's nauty package [20] via the GAP package GRAPE [26].

There is a function to read a list of block designs in external representation XML format, making use of the XML parser in the GAP package GAPDoc [18].

From the list of blocks of a block design D on v points, the package can determine (almost) all of the non-statistical properties of that block design that are described by the external representation, and can then output D and these properties in

external representation XML format.

In the following subsections we describe some applications of the DESIGN package to research problems.

5.1 BIBDs with repeated blocks

D. A. Preece is compiling a catalogue of BIBDs with repeated blocks, with $r \leq 21$ and $\gcd(b, r, \lambda) = 1$ [21]. Surprisingly, there are quite a few parameter lists (v, b, r, k, λ) for which a simple BIBD is known to exist, but the existence of a BIBD with repeated blocks is unknown.

The DESIGN package is being used to help fill up Preece's catalogue with BIBDs with repeated blocks. Often, cyclic or 1-rotational such designs have been found. For example, up to isomorphism, there are exactly three BIBDs with repeated blocks and $(v, b, r, k, \lambda) = (19, 57, 18, 6, 5)$ invariant under a Frobenius group of order 57. There is at least one 1-rotational BIBD with repeated blocks and $(v, b, r, k, \lambda) = (20, 76, 19, 5, 4)$. Many more examples can be found in [21].

5.2 Regular-graph designs

As part of the DTRS project we are generating equireplicate binary block designs with block size k that are A-, D-, E- or MV-optimal in the class of binary block designs with given (v, b, k). (These optimality criteria are described in Section 4.6.) In particular, we are checking and extending the influential work of John and Mitchell [17] by considering those designs with $v \leq 12$ and $r \leq 10$. Usually, but not always, the designs we seek can be shown to be regular-graph designs.

A *regular-graph design* $RGD(v, k; r)$ is a binary block design on v points, with blocks all of size k, such that every point is in exactly r blocks, and every pair of distinct points is contained in exactly λ or $\lambda + 1$ blocks, for some constant $\lambda > 0$ (which is necessarily $\lfloor r(k-1)/(v-1) \rfloor$). Such a regular-graph design D has an associated (regular) graph $\Gamma(D)$, whose vertex set consists of the points of D, with two distinct points joined by an edge precisely when they are contained in $\lambda + 1$ blocks of D.

The DESIGN package has been instrumental in discovering and classifying certain $RGD(v, k; r)$ D with given graph $\Gamma(D)$. In some difficult classifications we assume a specified subgroup of the automorphism group of each design we seek.

We now give an example application of the DESIGN package to the classification of regular-graph designs and their resolutions. We show the GAP and DESIGN commands used (at the GAP prompt `gap>`), followed by the GAP output if any (the use of `;;` after a command surpresses output). To start, the DESIGN package is loaded

into GAP.

```
gap> RequirePackage("design");
true
```

It can be shown that an $RGD(12, 4; 9)$ with associated graph Γ isomorphic to $K_{6,6}$ minus a 1-factor is A-, D- and E- optimal in the class of binary block designs with $(v, b, k) = (12, 27, 4)$.

```
gap> v:=12;;
gap> k:=4;;
gap> gamma_edges:=
>    [ [1,8],[1,9],[1,10],[1,11],[1,12],
>      [2,7],[2,9],[2,10],[2,11],[2,12],
>      [3,7],[3,8],[3,10],[3,11],[3,12],
>      [4,7],[4,8],[4,9],[4,11],[4,12],
>      [5,7],[5,8],[5,9],[5,10],[5,12],
>      [6,7],[6,8],[6,9],[6,10],[6,11] ];;
gap> gamma_nonedges:=Difference(Combinations([1..v],2),gamma_edges);;
```

We use the `BlockDesigns` function of the DESIGN package to classify (up to isomorphism) the $RGD(12, 4; 9)$ with associated graph Γ, such that each of these designs is invariant under $H = \langle (1, 2, 3)(4, 5, 6)(7, 8, 9)(10, 11, 12) \rangle$.

```
gap> H:=Group((1,2,3)(4,5,6)(7,8,9)(10,11,12));;
gap> designs:=BlockDesigns( rec(
>    v:=v, blockSizes:=[k],
>    tSubsetStructure:=rec(
>       t:=2, partition:=[gamma_edges,gamma_nonedges],
>             lambdas:=[3,2] ),
>    requiredAutSubgroup:=H ) );;
gap> Length(designs);
517
gap> Collected(List(designs,D->Size(AutGroupBlockDesign(D))));
[ [ 3, 496 ], [ 6, 20 ], [ 12, 1 ] ]
```

There are exactly 517 such designs up to isomorphism. Of these, 496 have automorphism group H, 20 have automorphism group of size 6, and one has automorphism group of size 12.

We next use the function `PartitionsIntoBlockDesigns` to classify the resolutions of these 517 designs.

```
gap> resolutions:=List( designs,
>    D->PartitionsIntoBlockDesigns( rec(
>       blockDesign:=D, v:=v, blockSizes:=[k],
>       tSubsetStructure:=rec( t:=1, lambdas:=[1] ) ) ) );;
gap> Collected(List(resolutions,Length));
```

```
[ [ 0, 515 ], [ 1, 2 ] ]
```

Just two of the designs are resolvable.

```
gap> I:=Filtered([1..Length(resolutions)],i->resolutions[i]<>[]);;
gap> List(designs{I},D->Size(AutGroupBlockDesign(D)));
[ 6, 3 ]
gap> List(resolutions{I},R->Size(R[1].autGroup));
[ 6, 3 ]
```

In each case there is just one resolution, fixed by the full automorphism group of the design. The first such resolution is:

```
gap> List(resolutions[I[1]][1].partition,D->D.blocks);
[ [ [ 1, 2, 4, 7 ], [ 3, 8, 9, 11 ], [ 5, 6, 10, 12 ] ],
  [ [ 1, 2, 11, 12 ], [ 3, 5, 7, 10 ], [ 4, 6, 8, 9 ] ],
  [ [ 1, 3, 6, 9 ], [ 2, 7, 8, 10 ], [ 4, 5, 11, 12 ] ],
  [ [ 1, 3, 10, 11 ], [ 2, 4, 9, 12 ], [ 5, 6, 7, 8 ] ],
  [ [ 1, 4, 8, 10 ], [ 2, 5, 9, 11 ], [ 3, 6, 7, 12 ] ],
  [ [ 1, 5, 8, 12 ], [ 2, 6, 9, 10 ], [ 3, 4, 7, 11 ] ],
  [ [ 1, 5, 9, 10 ], [ 2, 6, 7, 11 ], [ 3, 4, 8, 12 ] ],
  [ [ 1, 6, 8, 11 ], [ 2, 3, 10, 12 ], [ 4, 5, 7, 9 ] ],
  [ [ 1, 7, 9, 12 ], [ 2, 3, 5, 8 ], [ 4, 6, 10, 11 ] ] ]
```

Its automorphism group is:

```
gap> resolutions[I[1]][1].autGroup;
Group([ (1,3,2)(4,6,5)(7,9,8)(10,12,11),
        (1,7)(2,9)(3,8)(4,12)(5,11)(6,10) ])
```

The total run-time for this example computation is about ten minutes on a fast PC running Linux.

5.3 Latin squares

A Latin square can be encoded as a block design. Let R, C, and S be the sets of rows, columns, and symbols of a Latin square of order n; we assume that these three sets are pairwise disjoint. Now the point set of the design is $R \cup C \cup S$; for each cell of the square, there is a block $\{r, c, s\}$, where the cell lies in row r and column c and contains symbol s. This design is binary, has constant block size 3, and is equireplicate with replication number n; it has $3n$ points and n^2 blocks. This is the dual of the lattice design associated with the Latin square in Section 2.1.

Now a transversal of the Latin square is a set of n cells, with one in each row, one in each column, and one containing each symbol. The corresponding blocks of the design form a parallel class or 1-factor. An orthogonal mate of the square can be

regarded as a partition of the set of cells into n pairwise disjoint transversals; this partition is a resolution of the block design.

Jacobson and Matthews [16] described a Markov chain algorithm for choosing a random Latin square. This algorithm has been implemented, and we have begun using it to explore how many transversals, or orthogonal mates, a typical Latin square has. In particular, Ryser conjectured that any Latin square of odd order has a transversal. We have tested this conjecture on random Latin squares, without so far finding a counterexample. In this work, the DESIGN package is invoked for finding parallel classes or resolutions of the corresponding block design.

6 Documentation

In the DTRS library, the main item at present is the *Encyclopaedia of Design Theory* [8].

The Encyclopaedia contains descriptions of various classes of designs, under twelve headings: general description; structure as set of partitions, an incidence structure, or an array; use in experimental design; related designs; mathematical and statistical properties; cross-reference to the database; external references; miscellanea; and bibliography. (Not all of these headings are relevant for some designs.) In addition, there is a glossary, a bibliography, and links to historical material, as well as specific references to experimental design.

The library also contains the External Representation specification and its documentation, and a preprint collection.

7 Where next?

We would like your feedback on what we have done. Have we omitted something crucial? Send comments to `info@designtheory.org`.

We intend to extend this specification, first to handle non-binary block designs, and then to handle more general types of designs (row-column designs, factorial designs, semi-Latin squares, etc.)

We are also producing software to make it easy to read and write correctly specified designs, and to interface with combinatorial and statistical software (in the first instance GAP [13], MAGMA [10], Python [22], and R [23]).

References

[1] R. A. Bailey, Suprema and infima of association schemes, *Discrete Math.* **248** (2002), 1–16.

[2] R. A. Bailey and Peter J. Cameron, What is a design? And how should we classify them?, in preparation

[3] E. Bannai, An introduction to association schemes, *Methods of Discrete Mathematics* (S. Löwe, F. Mazzocca, N. Melone and U. Ott, eds.), Quaderni di Mathematica **5**, Seconda Università di Napoli, Napoli, 1999, pp. 1–70.

[4] T. Beth, D. Jungnickel and H. Lenz, *Design Theory* (2nd edition), Cambridge University Press, Cambridge, 1999.

[5] R. C. Bose and K. R. Nair, Partially balanced incomplete block designs, *Sankhyā* **4** (1939), 337–372.

[6] F. Buekenhout, Diagrams for geometries and groups, *J. Combinatorial Theory* (A) **27** (1979), 121–151.

[7] T. Caliński and S. Kageyama, *Block Designs: A Randomization Approach, Volume II: Design*, Springer-Verlag, 2003.

[8] Peter J. Cameron (editor), *Encyclopaedia of Design Theory*, `http://designtheory.org/library/encyc/`

[9] Peter J. Cameron, Peter Dobcsányi, John P. Morgan, Leonard H. Soicher, The External Representation of Block Designs, `http://designtheory.org/library/extrep/`

[10] J. Cannon and C. Playoust, *An Introduction to MAGMA*, University of Sydney, Sydney, Australia, 1993.

[11] Ph. Delsarte, An algebraic approach to the association schemes of coding theory, *Philips Research Reports Suppl.* **10** (1973).

[12] S. Evdokimov and I. Ponomarenko, Separability number and Schurity number of coherent configurations, *Electronic J. Combinatorics* **7**(1) (2000), #R31, `http://www.combinatorics.org`

[13] The GAP Group, GAP — Groups, Algorithms, and Programming, Version 4.3; Aachen, St Andrews, 2002, `http://www.gap-system.org/`

[14] A. Hanaki and I. Miyamoto, Classification of association schemes with small vertices, `http://kissme.shinshu-u.ac.jp/as/`

[15] D. G. Higman, *Combinatorial Considerations about Permutation Groups*, Mathematical Institute, Oxford, 1971.

[16] M. T. Jacobson and P. Matthews, Generating uniformly distributed random Latin squares, *J. Combinatorial Design* **4** (1996), 405–437.

[17] J. A. John and T. J. Mitchell, Optimal incomplete block designs, *J. Royal Statistical Society, Series B* **39** (1977), 39–43.

[18] F. Lübeck and M. Neunhöffer, The GAPDoc package for GAP, `http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/`

[19] J. D. Malley, *Optimal Unbiased Estimation of Variance Components*, Lecture Notes in Statistics **39**, Springer–Verlag, Berlin, 1986.

[20] B. D. McKay, `nauty`, `http://cs.anu.edu.au/people/bdm/nauty/`

[21] D. A. Preece, A selection of BIBDs with repeated blocks, $r \leq 21$, $\gcd(b, r, \lambda) = 1$, preprint, 2003.

[22] The Python programming language, `http://www.python.org/`

[23] The R Project for Statistical Computing, `http://www.r-project.org/`

[24] Relax NG Schema Language for XML, `http://relaxng.org/`

[25] K. R. Shah and B. K. Sinha, *Theory of Optimal Designs*, Springer, New York, 1989.

[26] Leonard H. Soicher, The GRAPE package for GAP, `http://www.maths.qmul.ac.uk/~leonard/grape/`

[27] J. Tits, *Buildings of Spherical Type and Finite BN-Pairs*, Lecture Notes in Math. **382**, Springer-Verlag, Berlin, 1974.