Abstract  Factorial designs are often used in various industrial and sociological experiments to identify significant factors and factor combinations that may affect the process response. In the statistics literature, several studies have investigated the analysis, construction, and isomorphism of factorial and fractional factorial designs. When there are multiple choices for a design, it is helpful to have an easy-to-use tool for identifying which designs are distinct, and which of those can be efficiently analyzed/has good theoretical properties. For this task, we present an \textit{R} library called \texttt{IsoCheck} that checks the isomorphism of multi-stage $2^n$ factorial experiments with randomization restrictions. Through representing the factors and their combinations as a finite projective geometry, \texttt{IsoCheck} recasts the problem of searching over all possible relabeling as a search over collineations, then exploits projective geometric properties of the space to make the search much more efficient. Furthermore, a bitstring representation of the factorial effects is used to characterize all possible rearrangements of designs, thus facilitating quick comparisons after relabeling. This paper presents several detailed examples with \textit{R} codes that illustrate the usage of the main functions in \texttt{IsoCheck}. Besides checking equivalence and isomorphism of $2^n$ multi-stage factorial designs, we demonstrate how the functions of the package can be used to create a catalog of all non-isomorphic designs, and find good designs as per a suitably defined ranking criterion.

Keywords  Projective geometry, Spreads, Collineation, Bitstring Mapping, Design catalog

AMS 2010 subject classifications: 62K15, 51A40, 51E14, 51E23, 05-04

DOI: 10.19139/soic-2310-5070-1494

1. Introduction

Factorial experiments are commonly used for significance assessment of factors in a wide spectrum of scientific and industrial applications including agriculture, engineering, and sociological phenomena. In many such experiments, complete randomization of the trials happens to be impractical for a variety of reasons. For instance, changing some factors can be prohibitively expensive, the trials may need to be partitioned into homogeneous batches at each stage, or the experimental units may need to be processed repeatedly in batches. To address these issues, randomization restrictions are imposed on the experimental trials. Classical designs for such factorial experiments include block designs, nested designs, split-plot designs, strip-plot designs, split-lot designs, and combinations thereof. A plethora of research articles have been published till date on the development of theoretical results and efficient algorithms to construct and analyze such designs. Some landmark papers can be traced back to the exploration of completely randomized designs and randomized block designs by Fisher (1942) and Yates (1937).
Later, experimenters investigated several factorial designs with multi-stage randomization restrictions. For instance, Addelman (1964), Bingham and Sitter (1999a) and many others investigated the split-plot designs; Miller (1997) developed the strip-plot designs for measuring wrinkles in a laundry experiment; blocked designs were investigated by Bisgaard (1994); Mee and Bates (1998), Butler (2004) and Bingham et al. (2008) presented some fundamental results on split-lot designs for analyzing the fabrication of integrated circuits using silicon wafers. The following books provide a detailed reference list Dean and Voss (1999), Mukerjee and Wu (2006), Wu and Hamada (2009), Hedayat et al. (2012), and Cheng (2016).

One important aspect in this area of research focuses on finding distinct designs that have useful theoretical properties and are easy to analyze. An essential task in such endeavors is to check for the isomorphism between different designs. In a completely randomized setup, two designs $d_1$ and $d_2$ are said to be isomorphic if $d_1$ can be obtained by applying relabeling and rearrangement of the factors and factor combinations on $d_2$. Franklin and Bailey (1977) developed theoretical results and algorithms for fractional factorial completely randomized designs. For factorial designs with randomization restrictions, isomorphic designs allow restricted rearrangement of the factorial effects within the subspace that characterizes the randomization defining factors. Bingham and Sitter (1999a) extended the results of Franklin and Bailey (1977) for split-plot designs. In a multi-stage factorial design with randomization restrictions, there are additional restrictions on the within block and between block relabeling and rearrangement (see Spencer et al. (2019)). Several researchers have also focused on proposing innovative design ranking criteria to sort through these non-isomorphic designs and create catalogs of good designs for ready usage by practitioners. Popular design ranking criteria are maximum resolution, maximize the number of clear 2-factor interactions, minimum aberration, etc. (Wu and Hamada, 2009).

Over the years, a few methodologies have been developed that unify different factorial designs with randomization restrictions. Nelder (1965a,b) developed a unified theory to analyze such experiments through simple block structures. This idea was further generalized to orthogonal block structures through association schemes by Speed and Bailey (1982) and Tjur (1984). More details can be found in Bailey (2004) and Cheng and Tsai (2011). Bingham et al. (2008) introduced the idea of randomization defining factors, which was further taken-up by Ranjan et al. (2009) and Ranjan et al. (2010) to develop Projective Geometry (PG) formulation based on the overlapping pattern of the randomization defining contrast subspaces (referred to as RDCSSs).

The key idea behind this RDCSS-based unified theory is to realize that the set of all factorial effects of a $q^n$ factorial design constitute an $(n - 1)$-dimensional finite projective geometry $P_n := PG(n - 1, q)$ over the Galois Field $GF(q)$. The randomization restrictions can be characterized by a projective subspace of $P_n$. Such multi-stage factorial designs have been constructed using two geometric structures called “spread” and “star”. The former partitions $P_n$ into disjoint RDCSSs, whereas the latter is a set of RDCSSs that forms a cover of $P_n$ with a common overlap. Ranjan et al. have also established that the construction and analysis of classical factorial designs with randomization restrictions like split-plot, strip-plot, split-lot, etc., can be derived from carefully chosen sets of RDCSSs of $P_n$. Recently, Spencer et al. (2019) developed isomorphism check algorithms for 2$^n$ multi-stage factorial designs based on spreads and stars. They presented a bitstring representation of $P_n$ which helps in developing an efficient search algorithm exploiting the geometric structure of RDCSSs to significantly reduce the search space. The results from projective geometry (e.g. Topaloca and Zhelezova (2010); Mateva and Topalova (2009)) are also applied to completely classify the isomorphism properties for several RDCSS-based designs (with small run sizes) that are useful from a practical standpoint.

The key contributions of this paper are highlighted as follows. We present an $R$ (R Core Team, 2020) package IsoCheck which assumes the RDCSS-based structure of the factorial designs and efficiently implements the algorithms of Spencer et al. (2019) to check the isomorphism. The key functions (a)
generate a relabelling and rearrangement of a given spread/star of $P_n$ via a proper collineation matrix; (b) check equivalence of two spreads (stars); (c) check isomorphism between two spreads (stars); and (d) find a spread embedded in the star-based design. Although, preliminary implementation of some functions can be found on the Github submission (as a part of Spencer et al. paper), IsoCheck presents robust and computationally more efficient version of these algorithms with several additional functions which are crucial for isomorphism check. In particular, the new functions are `is.spread()`, `is.star()`, `vectortostring()` and `stringtovector()`. IsoCheck is a free software and distributed under the General Public License and available from the Comprehensive R Archive Network. In this paper, we also demonstrate how the functions of IsoCheck can be used to construct a catalog of all non-isomorphic designs, and find good designs based on some suitably defined ranking criterion. Of course, it assumes that one can list out all possible designs with a given specification (i.e., number of factors, number of stages, and sizes of each RDCSS).

Section 2 of the paper provides necessary theoretical background on factorial designs, foundations of projective geometry $P_n = PG(n - 1, 2)$, including balanced spreads and covering stars of $P_n$, and algorithms for isomorphism check (developed by Spencer et al. (2019)). Section 3 outlines the usage and arguments of important functions of IsoCheck including the new functions that were not available in the Github submission. Section 4 provides several examples to present detailed illustration of how isomorphism check can be performed for both spread and star based designs. Section 5 outlines the catalog construction steps using IsoCheck for a few designs with small values of $n$. Section 6 concludes the article with a few remarks.

2. Methodology

In a $2^n$ factorial experiment, suppose the $n$ basic factors (main effects), $F_i$, $i = 1, \ldots, n$, are expressed as the $n$ canonical vectors, where $F_i$ has $n - 1$ zeros and exactly one 1 at the $i$-th location. These $n$ canonical vectors along with their interactions constitute an $n$-dimensional vector space $V$ over $GF(2)$. The set of all non-null elements of $V$ form a $(n - 1)$-dimensional projective space over $GF(2)$, denoted by $P_n := PG(n - 1, 2)$ (Bose, 1947). For a given stage of a $n$-factor multi-stage / multi-stratum factorial design (e.g., split-plot, strip-plot, split-lot, etc.), the randomization restrictions can be characterized by a projective subspace of $P_n$. In the context of projective geometry, such a subspace is called a flat. More formally, a $(t - 1)$-flat of $P_n$ is defined as the set of $2^t - 1$ non-null vectors spanned by $t$ linearly independent effects of $P_n$. In the design and analysis of experiments language, we refer to this $(t - 1)$-flat as a randomization defining contrast subspace (RDCSS) with $t$ linearly independent factors or factor interactions (ii), which is in the same spirit as fraction defining contrast subgroup (FDCS) used to characterize traditional fractional factorial designs (Bingham and Sitter, 1999a).

The multi-stage RDCSS-based designs proposed by Ranjan et al. (2009) and Ranjan et al. (2010) are primarily derived from two geometric structures called spreads and stars of $P_n$, which are essentially the set of distinct flats of $P_n$ that also form a cover of $P_n$. In a nutshell, a spread is a collection of disjoint flats, whereas a star is a set of distinct flats that allow a common overlap. Ranjan et al. proposed several multi-stage factorial designs by exploiting the sizes of flats and their overlaps. Recently, Spencer et al. (2019) has developed algorithms for checking isomorphism between a smaller class of balanced spread- and balanced star- based designs. The $R$ library (IsoCheck) presented here facilitates easy implementation of these isomorphism check algorithms.

**Definition 1:** For $1 \leq t \leq n$, a balanced $(t - 1)$-spread $\psi$ of $P_n := PG(n - 1, 2)$ is a set of $(t - 1)$-flats of $P_n$ which partitions $P_n$.  

In other words, a spread of \( P_n \) is a set of projective subspaces of \( P_n \) that forms a partition and a cover of \( P_n \), i.e., \( \psi \) consists of all disjoint flats that includes every element of \( P_n \). A \((t-1)\)-spread of \( P_n \) contains \( \mu = \frac{2^n - 1}{(2^t - 1)} \) distinct \((t-1)\)-flats that can be used to construct the RDCSSs for different stages of randomization. A necessary and sufficient condition for the existence of a balanced \((t-1)\)-spread is that \( t \) divides \( n \) (André, 1954).

If \( t \) does not divide \( n \), then either a partial \((t-1)\)-spread or an overlapping set of RDCSSs have to be used for design construction. Ranjan et al. (2009) presents some results on design construction using partial spreads. However, we restrict to the algorithms on the isomorphism of complete spread based designs. In cases when overlap between two \((t-1)\)-flats (i.e., RDCSSs) is unavoidable, we focus on balanced star-based designs in \( P_n \).

**Definition 2:** For \( 0 \leq t_0 < t < n \), a balanced star of \( P_n \) is a set of \( \mu \) \((t-1)\)-flats and a \((t_0-1)\)-flat in \( P_n \), such that the intersection of any two of the \( \mu \) flats is the \((t_0-1)\)-flat. Such a star, denoted by \( \Omega = St(n, \mu, t, t_0) \), consists of \( \mu \) distinct \((t-1)\)-flats called rays and a \((t_0-1)\)-flat known as the nucleus. The covering star \( St(n, \mu, t, t_0) \) provides \( \mu = \frac{2^n - t_0 - 1}{2^t - t_0 - 1} \) overlapping RDCSSs of size \( 2^t - 1 \) each.

A balanced spread can be constructed using the cyclic approach of Hirschfeld (1998). The non-zero elements of \( GF(2^n) \) can be written as \( \{\omega^0, \omega^1, \ldots, \omega^{2^n-2}\} \), where \( \omega \) is a primitive element, and \( \omega^i = \alpha_0 \omega^0 + \alpha_1 \omega^1 + \cdots + \alpha_{n-1} \omega^{n-1} \), for \( 0 \leq i \leq 2^n - 2 \), correspond to the vector representation \( (\alpha_0, \ldots, \alpha_{n-1}) \) of elements in \( P_n \). Let \( t \) be an integer where \( t \) divides \( n \). If we want to construct a \((t-1)\)-spread \( \psi \) of \( P_n \), we write the non-zero elements of \( GF(2^n) \) in cycles of length \( \mu \), where \( \mu = \frac{2^n - 1}{(2^t - 1)} \). Hirschfeld (1998) showed that the \( f_i \)'s are \((t-1)\)-flats as shown in Table 1 and \( \psi = \{f_1, \ldots, f_\mu\} \) partitions the set of all non-zero elements of \( GF(2^n) \), i.e., \( \psi \) is a \((t-1)\)-spread of \( P_n \).

| Table 1. The elements of \( GF(2^n) \) in cycles of length \( \mu \) |
|----------------|----------------|----------------|----------------|----------------|
| \( f_1 \)     | \( f_2 \)     | \( \ldots \)  | \( f_\mu \)   |
| \( \omega^0 \) | \( \omega^1 \) | \( \ldots \)  | \( \omega^{\mu-1} \) |
| \( \omega^\mu \) | \( \omega^{\mu+1} \) | \( \ldots \)  | \( \omega^{2\mu-1} \) |
| \( \vdots \)  | \( \vdots \)  | \( \ddots \)  | \( \vdots \)   |
| \( \omega^{2^n-\mu-1} \) | \( \omega^{2^n-\mu} \) | \( \ldots \)  | \( \omega^{2^n-2} \) |

As a quick example, Table 2 shows a cyclic spread of \( P_4 = PG(3,2) \) using the primitive polynomial \( \omega^4 + \omega + 1 \). One can use these five 1-flats (subspaces generated using two independent factors or \( f_i \)'s each) for constructing up to five RDCSSs of a multi-stage factorial design with 4 two-level factors.

| Table 2. A 1-spread of \( PG(3,2) \) obtained via the primitive polynomial \( \omega^4 + \omega + 1 \). |
|----------------|----------------|----------------|----------------|----------------|
| \( f_1 \)     | \( f_2 \)     | \( f_3 \)     | \( f_4 \)     | \( f_5 \)     |
| D              | C              | B              | A              | CD             |
| BC             | AB             | ACD            | BD             | AC             |
| BCD            | ABC            | ABCD           | ABD            | AD             |
We can also construct a balanced covering star $\Omega = St(n, \mu, t, t_0)$ using such cyclic spreads as shown in Table 1. Let $\{f_1, \ldots, f_\mu\}$ denote the flats consisting of a $t_0 \times (t - t_0 - 1)$-spread $\psi$ of $\mathcal{P}_{n-t_0}$. Then there exists a $(t_0 - 1)$-flat $\pi$ in $\mathcal{P}_n \setminus \mathcal{P}_{n-t_0}$ such that $f_i^* = (f_i, \pi)$ and $\{f_1^*, \ldots, f_\mu^*\}$ form a covering star $\Omega$ of $\mathcal{P}_n$. We denote such stars as $\Omega = \psi \times \pi$ (see Ranjan et al. (2010) for the proof of this result).

We now return to the necessary theoretical details for discussing isomorphism between two designs $d_1$ and $d_2$. Checking isomorphism between two multi-stage factorial designs requires essentially two major checks: (a) rearrangement of factorial effects within each RDCSS and the rearrangement / renaming of RDCSSs themselves, and (b) relabelling of factorial effects. The first step simply amounts to checking the equality of $d_1$ and $d_2$ as sets, and if satisfied, we refer to $d_1$ and $d_2$ as equivalent.

**Definition 3:** Let $d_1$ and $d_2$ be two $2^n$ multi-stage factorial designs with $\mu$ stages of randomization, and let the respective sets of RDCSSs be denoted by $d_1 = \{f_1, f_2, \ldots, f_\mu\}$ and $d_2 = \{g_1, g_2, \ldots, g_\mu\}$. Then $d_1$ is said to be equivalent to $d_2$ (denoted by $d_1 \equiv d_2$) if and only if, for every $f_i \in d_1$, there is a unique $g_j \in d_2$ such that $\{f_i\} = \{g_j\}$ (set equality), for $1 \leq i, j \leq \mu$.

Let $\mathcal{E}(d_1)$ denote the equivalence class of $d_1$, and if $|f_i| = 2^t - 1$ for $f_i \in d_1$, then, $|\mathcal{E}(d_1)| = \mu!(2^t - 1)!^\mu$. For instance, the size of equivalence class for the design presented in Table 2 is $5!((3!)^5) = 933120$. When $n$ is large, checking the equivalence of two designs by naively iterating through the entire equivalence class is too computationally intensive. However, the computational burden can be reduced through a combination of sorting and bitstring representation scheme.

Every element of $\mathcal{P}_n$ can be represented as a unique binary string of $2^n - 1$ bits with exactly one non-zero entry. In this representation, the contents of any RDCSS $f$ can now be identified by the sum of the bitstring representations of its elements. For example, following the Yates Order (Box et al., 1978) of $\mathcal{P}_3$, the bitstring representations of the elements of $\mathcal{P}_3 = \{A, B, AB, C, AC, BC, ABC\}$ are $A \rightarrow 1000000$, $B \rightarrow 0100000$, $\ldots$, $ABC \rightarrow 0000001$. Then, $f = \{AB, AC, BC\}$ can be uniquely represented by the bitstring representation 0010110. As a result, the equivalence check between two designs $d_1$ and $d_2$ simplifies to only comparing two sets of $\mu$ bitstrings, which is much cheaper than having to compare $d_1$ to all possible rearrangements of $d_2$.

The second major step towards isomorphism check is to go through all possible relabelings of the factors and factor combinations in $d_1$ and then check for equivalence between the relabeled $d_1$ and $d_2$. In projective geometry literature, relabeling is typically done by applying a collineation.

**Definition 4:** A collineation of $\mathcal{P}_n = PG(n - 1, 2)$ is a mapping of the points from $\mathcal{P}_n$ to $\mathcal{P}_n$ such that $(t - 1)$-flats gets mapped to $(t - 1)$-flats for all $1 \leq t \leq n$.

A collineation of $\mathcal{P}_n$ can be characterized by an $n \times n$ matrix $\mathcal{C}$ over $GF(2)$, referred to as the collineation matrix (Batten, 1997), where the $j$-th column of $\mathcal{C}$ is the image of the basic factor $F_j$ (i.e., the effect in $\mathcal{P}_n$ that $F_j$ gets mapped to). For instance, a $3 \times 3$ collineation matrix for $PG(2, 2)$ is

$$
\mathcal{C} = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
$$

which relabels the basic factors as $A \rightarrow A$, $B \rightarrow B$ and $C \rightarrow BC$. 

**Definition 5:** Two $2^n$ RDCSS-based designs $d_1 = \{f_1, f_2, \ldots, f_\mu\}$ and $d_2 = \{g_1, g_2, \ldots, g_\mu\}$ are said to be isomorphic (denoted by $d_1 \cong d_2$) if and only if there exists a collineation $C$ over $P_n$ such that $C(d_1) \equiv d_2$. In this case, we say that $C$ is an isomorphism establishing collineation (IEC) from $d_1$ to $d_2$.

We now briefly describe the search algorithm for isomorphism check between two spread- (star-) based designs, proposed by Spencer et al. (2019).

### 2.1. Isomorphism Check for Spread-based Designs

For checking the isomorphism between two $(t-1)$-spread based multi-stage factorial designs characterized by $d_1 = \{f_1, f_2, \ldots, f_\mu\}$ and $d_2 = \{g_1, g_2, \ldots, g_\mu\}$ in $P_n$, we follow a systematic approach to search for the IEC by iterating through the candidate collineations. First, choose $l_0 = n/t$ out of $\mu$ RDCSSs in $d_1$ that can generate a basis $\{x_1, \ldots, x_n\}$ for $P_n$ such that each of the $l_0$ RDCSSs contributed $t$ independent points to the basis. Then, construct a collineation matrix $C_{x, B}$ to transform the $x$’s to a canonical basis with basic factors, $\{A, B, \ldots\}$. Note that the equivalence of $C_{x, B}(d_1)$ and $d_2$ implies the isomorphism between $d_1$ and $d_2$. Now, iterate through all possible sets of $l_0$ RDCSSs from $d_2$ to construct the basis set $\{y_1, \ldots, y_n\}$ in $d_2$ and the corresponding collineation matrix $C_{B, y}$ that maps the canonical basis to $y$’s. If $C_{B, y}(C_{x, B}(d_1))$ is equivalent to $d_2$, then we have found the IEC. The step-by-step algorithm from Spencer et al. (2019) is now presented for convenience of the readers.

#### Algorithm 1 Isomorphism check between two $(t-1)$-spread based designs in $P_n$

1: Choose $l_0 = n/t$ out of $\mu$ RDCSSs from $d_1 = \{f_1, f_2, \ldots, f_\mu\}$, such that $\langle \cup_{i=1}^{l_0} f_{u_i} \rangle = P_n$. Let these be $\{f_{u_1}, f_{u_2}, \ldots, f_{u_{l_0}}\}$ for $1 \leq u_1, u_2, \ldots, u_{l_0} \leq \mu$.
2: For $i = 1, \ldots, l_0$, specify $\{x_{i,1}, \ldots, x_{i,t}\} \in f_{u_i}$ such that $f_{u_i} = \{x_{i,1}, \ldots, x_{i,t}\}$.
3: Construct the collineation matrix $C_{x, B}$ which maps the $x$’s in Step 2 to the canonical basis $\{A, B, \ldots\}$ such that each $x_{i,j}$ is mapped to $F_{i((i-1))+j}$.
4: (a) Choose $l_0$ out of $\mu$ RDCSSs from $d_2 = \{g_1, g_2, \ldots, g_\mu\}$ for mapping $f_{u_i}$’s (note that ordering is important). Let that be $\{g_{v_1}, g_{v_2}, \ldots, g_{v_{l_0}}\}$ for $1 \leq v_1, v_2, \ldots, v_{l_0} \leq \mu$. If $|\langle g_{v_1} \cup \ldots \cup g_{v_{l_0}} \rangle| < 2^n - 1$, proceed to the next choice for $\{g_{v_1}, g_{v_2}, \ldots, g_{v_{l_0}}\}$.
(b) For $i = 1, \ldots, l_0$, choose a $\{y_{i,1}, \ldots, y_{i,t}\} \in g_{v_i}$ such that $g_{v_i} = \{y_{i,1}, \ldots, y_{i,t}\}$.
(c) Choose one of the $(l_0)!$ permutations of $1, \ldots, l_0$, say $\sigma_k$, for $k = 1, \ldots, (l_0)!$.
(d) Construct $C_{B, y}$ which maps the canonical basis elements to $y$’s such that $F_{i((i-1))+j}$ is mapped to $g_{v_{k(i,j)}}$.
(e) If $C_{B, y}(C_{x, B}(d_1))$ is equivalent to $d_2$, then $d_1 \cong d_2$, and report $C = C_{B, y}C_{x, B}$ as an IEC and exit; otherwise, continue.
(f) Go to Step 4(c) and choose another ordering $\sigma_k$ if possible, otherwise, continue.
(g) Go to Step 4(b) and choose another basis if possible, otherwise, continue.
(h) Go to Step 4(a) and choose another set of RDCSSs if possible, otherwise report that $d_1$ and $d_2$ are non-isomorphic.

### 2.2. Isomorphism check for star based design

A balanced star $\Omega = St(n, \mu, t, t_0)$ can be expressed as $\Omega = \psi \times \pi$, where $\psi$ is a $(t-t_0)-1$-spread of $P_{n-t_0}$, and $\pi$ is $(t_0-1)$-dimensional subspace in $P_n \setminus P_{n-t_0}$. Thus, the isomorphism check between two star-based designs $d_1$ and $d_2$ (with $(t_0-1)$-dimensional nuclei) can be reduced to checking isomorphism between two $(h-1)$-spreads of $P_{n-t_0}$ by iterating through the elements of $C_{n-t_0}$ instead of $C_n$.
2 (presented by Spencer et al. (2019)) summarizes the steps of how Algorithm 1 can be used to search for an IEC between $\psi_1$ and $\psi_2$ assuming $\Omega_i = \psi_i \times \pi_i$ for $i = 1, 2$.

**Algorithm 2** Isomorphism check between two $St(n, \mu, t, t_0)$-based designs $d_1$ and $d_2$, which correspond to stars $\Omega_1 = \psi_1 \times \pi_1$ and $\Omega_2 = \psi_2 \times \pi_2$, respectively.

1. Determine two bases $\{p_{1,1}, \ldots, p_{1,t_0}\}$ and $\{p_{2,1}, \ldots, p_{2,t_0}\}$ of nuclei $\pi_1$ and $\pi_2$, respectively.
2. Construct a collineation matrix $C_{\pi_1, B}$ mapping $\{p_{1,1}, \ldots, p_{1,t_0}\}$ to the $t_0$ trailing basic factors $F_{n-t_0+1}, \ldots, F_n$. The pre-images of $F_1, \ldots, F_{n-t_0}$ can be chosen as an arbitrary linearly independent set from $P_n \setminus \pi_1$.
3. Similarly, construct a collineation matrix $C_{\pi_2, B}$ mapping $\{p_{2,1}, \ldots, p_{2,t_0}\}$ to the $t_0$ trailing basic factors $F_{n-t_0+1}, \ldots, F_n$.
4. Extract designs $d_1^*$ and $d_2^*$ on $P_{n-t_0}$ corresponding to the spreads $C_{\pi_1, B(\psi_1)}$ and $C_{\pi_2, B(\psi_2)}$.
5. Run Algorithm 1 on $d_1^*$ and $d_2^*$. If we come across a $C^*$ that is an IEC, then an IEC for $d_1$ and $d_2$ is given by $C = C_{\pi_2, B}^{-1} \circ C^* \circ C_{\pi_1, B}$. Otherwise, $d_1$ and $d_2$ are non-isomorphic.

3. IsoCheck package

In this section, we examine the functions in the IsoCheck package which checks the isomorphism for RDCSS-based designs through the theoretical results and efficient algorithms proposed by Spencer et al. (2019). We will discuss both spread-based and star-based designs. The most important functions are `checkSpreadIsomorphism()` and `checkStarIsomorphism()` which take two spreads and stars, respectively, as inputs and return a Boolean (TRUE/FALSE) indicating whether or not the inputs are isomorphic. If they are isomorphic, then these two functions also return isomorphism establishing collineations (IECs). We have also packaged a few important additional functions for efficiently checking the equivalence and iterating through different collineations. All of these functions have been summarized as follows:

- **checkSpreadIsomorphism()**: requires two spreads (`spread1` and `spread2`) as inputs whose isomorphism is to be checked. It also takes an argument called `returnfirstIEC` which indicates whether all isomorphism establishing collineations should be returned (default), or terminate only after the first one is found.

- **checkStarIsomorphism()**: works in the similar way as `checkSpreadIsomorphism()`, except it requires two stars (`star1` and `star2`) as inputs, and uses Algorithm 2.

- **checkSpreadEquivalence()**: takes two spreads (`spread1` and `spread2`) as inputs, and compares their sorted bitstring representations. It returns a Boolean- TRUE if the spreads are equivalent, else it returns FALSE.

- **checkStarEquivalence()**: works in the similar way as `checkSpreadEquivalence()`, except it requires two stars (`star1` and `star2`) as inputs.

- **getBitstrings()**: takes a spread (or a star) and returns its bitstring representation. This makes the equivalence check, and subsequently the isomorphism check, significantly more computationally efficient as compared to the naive approach.

- **applyCollineation()**: takes an $n \times n$ collineation matrix and a spread (or a star), and produces the relabeled spread (or the star).
star_to_spread() : takes a star star (assumes the structure to be \( \Omega = \psi \times \pi \)), isolates the nucleus \( \pi \) from this star by using a suitable collineation matrix, and returns the spread \( \psi \) embedded in this star along with the collineation matrix.

We have also included two more functions called stringtovector() and vectortostring() in the \( R \) package IsoCheck, which facilitates the users in inputting the spreads and stars. However, for quick usage and illustration, several spreads and stars have also been hard-coded, e.g., spreadn4t2a, spreadn4t2b, starn8t5a, starn8t5b (see the package manual for other builtin spreads and stars). These functions are illustrated in details in Section 4.

The function star_to_spread() is called within checkStarIsomorphism() which separates the nucleus \( \pi_i \) and \( 1(\tau - \tau_0) - 1\)-spread \( \psi_i \) of \( \mathcal{P}_4 \) from the star \( \Omega_i \) where \( i = 1, 2 \) such that \( \Omega_1 = \psi_1 \times \pi_1 \) and \( \Omega_2 = \psi_2 \times \pi_2 \). Then the problem is reduced to checking the isomorphism between the two spreads \( \psi_1 \) and \( \psi_2 \) and equality of two nuclei \( \pi_1 \) and \( \pi_2 \). This is Algorithm 2 described in detail by Spencer et al. (2019). See Example 4 for an illustration.

4. Examples using IsoCheck

This section demonstrates the usage of different functions in IsoCheck for checking equivalence and isomorphism of multi-stage factorial designs with randomization restrictions. We also show how some of these functions can be used in design ranking and further creating catalogs of good non-isomorphic designs for a practitioner.

Example 1 We know from André (1954) that there exists non-trivial 1-spreads \( (t = 2) \) of \( \mathcal{P}_4 \) and Soicher (2000) has shown that they are isomorphic. Here we illustrate how isomorphism can be established between two 1-spreads of \( \mathcal{P}_4 \). Table 3 shows the two 1-spreads of \( \mathcal{P}_4 \).

<table>
<thead>
<tr>
<th>spreadn4t2a</th>
<th>spreadn4t2b</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( g_1 )</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>ABC</td>
</tr>
<tr>
<td>CD</td>
<td>AC</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( g_2 )</td>
</tr>
<tr>
<td>BC</td>
<td>CD</td>
</tr>
<tr>
<td>AB</td>
<td>ABCD</td>
</tr>
<tr>
<td>ACD</td>
<td>B</td>
</tr>
<tr>
<td>BD</td>
<td>AD</td>
</tr>
<tr>
<td>AC</td>
<td>AB</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>( g_3 )</td>
</tr>
<tr>
<td>BCD</td>
<td>ACD</td>
</tr>
<tr>
<td>ABC</td>
<td>ABD</td>
</tr>
<tr>
<td>ACD</td>
<td>BD</td>
</tr>
<tr>
<td>ABCD</td>
<td>BCD</td>
</tr>
<tr>
<td>ABD</td>
<td>BC</td>
</tr>
<tr>
<td>AD</td>
<td>AB</td>
</tr>
<tr>
<td>AC</td>
<td>ABD</td>
</tr>
</tbody>
</table>

The \( R \) package IsoCheck includes these two 1-spreads of \( \mathcal{P}_4 \) that can be called using the data() command as follows:

R> data("spreadn4t2a", package = "IsoCheck")
R> data("spreadn4t2b", package = "IsoCheck")

The users can use stringtovector() to manually input the spreads of their choice as a three dimensional array \( spr[i, j, k] \) which specifies the presence of \( i \)-th basic factor in the \( j \)-th effect of the \( k \)-th flat of \( spr \). For instance, spreadn4t2a can be coded as follows.

R> spr <- array(NA,c(n=4,flatsize=3,mu=5))
R> spr[,1,1] <- stringtovector("D",n=4)
R> spr[,2,1] <- stringtovector("BC",n=4)
R> spr[,3,1] <- stringtovector("BCD",n=4)
...
R> spr[,3,5] <- stringtovector("AD",n=4)

We can check the isomorphism between the two spreads `spreadn4t2a` and `spreadn4t2b` (shown in Table 3) by calling `checkSpreadIsomorphism()`. This function implements Algorithm 1 described in Spencer et al. (2019) to search for the IEC by iterating through all the possible collineations over $P_4$.

R> test <- checkSpreadIsomorphism(spreadn4t2a, spreadn4t2b, + returnfirstIEC = FALSE, printstatement = TRUE)

The output printed on console is shown as follows:

```
[1] "percent done: 0.28"
[1] "percent done: 0.56"
...
[1] "percent done: 99.72"
[1] "percent done: 100"
[1] "The two spreads are isomorphic. For example, one isomorphism establishing collineation is"
[1,] 0 1 0 0
[2,] 1 0 0 0
[3,] 1 1 1 1
[4,] 1 0 0 1
```

The argument `printstatement` when set to `TRUE` declares the isomorphism at the end of the run and returns the first IEC. The output of the function `checkSpreadIsomorphism()` returns a Boolean value `TRUE` and enumerates all the isomorphism establishing collineations. The command `test$IECs[[1]]` shows the first IEC, and `length(test$IECs)` returns the total number of IECs found by the algorithm. The user can instead choose to terminate the search for IEC as soon as the first IEC is found and cut down the run time, by using the argument `returnfirstIEC = TRUE`.

R> test$result
[1] TRUE
R> test$IECs[[1]]
[1,] 0 1 0 0
[2,] 1 0 0 0
[3,] 1 1 1 1
[4,] 1 0 0 1

R> length(test$IECs)
[1] 360

In order to cross-verify that the resultant matrix `test$IEC[[1]]` indeed establishes isomorphism between the two spreads, we first apply the collineation matrix to the first spread `spreadn4t2a` and then check if the resulting spread `Cspreadn4t2a` is equivalent to the second spread `spreadn4t2b`.
R> IEC <- test$IECs[[1]]
R> Cspreadn4t2a <- applyCollineation(IEC, spreadn4t2a)
R> vectortostring(Cspreadn4t2a)

Table 4 presents the output of vectortostring(Cspreadn4t2a).

<table>
<thead>
<tr>
<th>Cspreadn4t2a</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1  h2  h3  h4  h5</td>
</tr>
<tr>
<td>CD  C   AC  BCD D</td>
</tr>
<tr>
<td>A   ABD BC  AD BD</td>
</tr>
<tr>
<td>ACD ABCD ABC  ABC  B</td>
</tr>
</tbody>
</table>

By visual inspection it is clear that spreadn4t2b is simply a rearrangement of Cspreadn4t2a, but for bigger problems it becomes difficult to conclude anything about equivalence by simple visual inspection. Hence we use checkSpreadEquivalence() function, which converts the spreads to their bitstring representations and the comparison becomes easy.

R> checkSpreadEquivalence(Cspreadn4t2a, spreadn4t2b)
[1] TRUE

**Example 2** We now consider the isomorphism of 2-spreads \( t = 3 \) of \( \mathcal{P}_6 = PG(5,2) \). For illustration, we choose two distinct built-in spreads, spreadn6t3a and spreadn6t3b, however, the users may input the spreads of their own choice.

R> checkSpreadEquivalence(spreadn6t3a, spreadn6t3b)
[1] FALSE

The two spreads spreadn6t3a and spreadn6t3b are presented in Tables 5 and 6 respectively.

<table>
<thead>
<tr>
<th>spreadn6t3a</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1  f2  f3  f4  f5  f6  f7  f8  f9</td>
</tr>
<tr>
<td>A   B   C   D   E   F   BD  AC  AD</td>
</tr>
<tr>
<td>EF  AF  AB  BC  CD  DE  BF  CE  BE</td>
</tr>
<tr>
<td>AEF ABF ABC BCD CDE DEF DF AE ABDE</td>
</tr>
<tr>
<td>BCE CDF ADE BEF ACF ABD ACE BDF CF</td>
</tr>
<tr>
<td>ABCE BCDF ACDE BDEF ACEF ABDF ABCDE ABCDF ACDF</td>
</tr>
<tr>
<td>BCF ACD BDE CEF ADF ABE ABCEF BCDEF BCEF</td>
</tr>
<tr>
<td>ABCF ABCD BCDE CDEF ADEF ABEF ACDEF ABDEF ABCDEF</td>
</tr>
</tbody>
</table>

We now proceed to check isomorphism between the two spreads.

Table 6. A 2-spread of $\mathcal{P}_6 = PG(5, 2)$

<table>
<thead>
<tr>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
<th>$g_5$</th>
<th>$g_6$</th>
<th>$g_7$</th>
<th>$g_8$</th>
<th>$g_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>E</td>
<td>DF</td>
<td>B</td>
<td>A</td>
<td>EF</td>
<td>BE</td>
<td>ABCDF</td>
<td>AC</td>
</tr>
<tr>
<td>AEF</td>
<td>ABCEF</td>
<td>ABCE</td>
<td>DEF</td>
<td>BDF</td>
<td>AB</td>
<td>F</td>
<td>ADF</td>
<td>AE</td>
</tr>
<tr>
<td>BCEF</td>
<td>ABCF</td>
<td>ABCDEF</td>
<td>BDEF</td>
<td>ABDF</td>
<td>ABEF</td>
<td>BEF</td>
<td>BC</td>
<td>CE</td>
</tr>
<tr>
<td>ADEF</td>
<td>BDE</td>
<td>C</td>
<td>AF</td>
<td>ABCDE</td>
<td>ACE</td>
<td>BCDF</td>
<td>BF</td>
<td>DE</td>
</tr>
<tr>
<td>BCDEF</td>
<td>BD</td>
<td>CDF</td>
<td>ABF</td>
<td>BCDE</td>
<td>ACF</td>
<td>CDEF</td>
<td>ACD</td>
<td>ACDE</td>
</tr>
<tr>
<td>D</td>
<td>ACDF</td>
<td>ABE</td>
<td>ADE</td>
<td>ACEF</td>
<td>BCE</td>
<td>BCD</td>
<td>ABD</td>
<td>AD</td>
</tr>
<tr>
<td>ABCD</td>
<td>ACDEF</td>
<td>ABDEF</td>
<td>ABDE</td>
<td>CEF</td>
<td>BCE</td>
<td>CDE</td>
<td>CF</td>
<td>CD</td>
</tr>
</tbody>
</table>

R> test <- checkSpreadIsomorphism(spread6t3a, spread6t3b,
+   returnfirstIEC = TRUE, printstatement = FALSE)

R> test$result

[1] TRUE

R> test$IECs

[[1]]
[,1] [ ,2] [ ,3] [ ,4] [ ,5] [ ,6]
[1,] 1 0 0 1 1 0
[2,] 0 1 0 1 0 0
[3,] 1 1 0 0 0 1
[4,] 1 1 1 0 0 1
[5,] 0 1 0 1 0 1
[6,] 1 1 1 1 1 0

Similar to Example 1, one can use `applyCollineation()` and `checkSpreadEquivalence()` to cross-check whether the collineation matrix found is indeed an IEC. Since we used "returnfirstIEC = TRUE", the code terminates after the first IEC was found.

**Example 3** Mee and Bates (1998) constructed and analyzed split-lot experimental designs for multistage batch processing for fabrication of integrated circuits. Here we illustrate the isomorphism between two designs for a 64-wafer experiment which involves six basic factors and nine disjoint RDCCSs, each of size seven. The two designs are $IC_1 = \{\langle A, EF, BCE \rangle, \langle B, AF, CDF \rangle, \langle C, AB, ADE \rangle, \langle D, BC, BEF \rangle, \langle E, CD, ACF \rangle, \langle F, DE, ABD \rangle, \langle BD, BF, ACE \rangle, \langle AC, CE, BDF \rangle, \langle AD, BE, CF \rangle\}$ and $IC_2 = \{\langle A, BD, CF \rangle, \langle B, AF, CE \rangle, \langle C, BF, DE \rangle, \langle D, AC, BE \rangle, \langle E, AB, DF \rangle, \langle F, AE, CD \rangle, \langle AD, BC, EF \rangle, \langle ACE, ADF, BEF \rangle, \langle ABC, ADE, CEF \rangle\}$. Here, $\langle \ldots \rangle$ denotes the span of the basis vectors within it. We can construct the full flats using these basis vectors with the following code.

R> basistoflat3 <- function(b1,b2,b3, n){
  +   base1 <- stringtovector(b1, n)
  +   base2 <- stringtovector(b2, n)
  +   base3 <- stringtovector(b3, n)
  +   v1 <- (base1 + base2) %%2
\begin{verbatim}
+ v2 <- (base1 + base3) %%2
+ v3 <- (base2 + base3) %%2
+ v4 <- (v1 + base3) %%2
+ return(cbind(base1, base2, v1, base3, v2, v3, v4))
+
R> n <- 6
R> flatsize <- 2^3-1
R> mu <- 9
R> spread_IC1 <- array(NA, c(n, flatsize, mu))
R> spread_IC1[,,1] <- basistoflat3("A", "EF", "BCE", n)
R> spread_IC1[,,2] <- basistoflat3("B", "AF", "CDF", n)
... 
R> spread_IC1[,,9] <- basistoflat3("AD", "BE", "CF", n)

The complete spread IC_1 is now constructed and stored in the array spread_IC1. Similarly, the second spread IC_2 is constructed and stored as spread_IC2. We can now proceed to check the isomorphism between spread_IC1 and spread_IC2.

R> test<- checkSpreadIsomorphism(spread_IC1, spread_IC2, + returnfirstIEC = T)
R
>

$\text{test}$

$\text{result}$

[1] TRUE

Hence, both the designs IC_1 and IC_2 are isomorphic. As in Example 2, we saved some computational time by terminating the algorithm after the first IEC was found. One can extract the IEC by calling test$IECs.

Example 4 We now demonstrate the isomorphism check between two stars starn5t3a and starn5t3b denoted by $\Omega_1$ and $\Omega_2$, respectively, (see Table 7). Both of these are balanced covering stars of $P_5 = PG(4,2)$ denoted by $St(5,5,3,1)$. The stars consist of $\mu = \frac{2^n - n - 1}{2^{n-t_0}-1} = \frac{5^5-1-1}{2^3-1} = 5$ rays of 2-flats overlapping with a nucleus which is a 0-flat in $P_5$. A close look at Table 7 shows that the nucleus in $\Omega_1$ is $\pi_1 = \langle A \rangle$ and in $\Omega_2$ it is $\pi_2 = \langle ABC \rangle$.

The following code can be used to quickly verify the equivalence of the two stars.

R> checkStarEquivalence(starn5t3a, starn5t3b)

[1] FALSE

As discussed in Sections 2, one can check for the isomorphism between two stars either by directly using checkStarIsomorphism() or by first decomposing the star to a spread via star_to_spread() and then use checkSpreadIsomorphism(). The first approach is implemented as follows.

R> checkStarIsomorphism(starn5t3a, starn5t3b, returnfirstIEC = TRUE)

$\text{$result$}$

[1] TRUE

$\text{IECs[[1]]}$
\end{verbatim}
Table 7. Two stars of $\mathcal{P}_5 = PG(4,2)$

<table>
<thead>
<tr>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
<th>$g_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>DE</td>
<td>ABC</td>
<td>AE</td>
<td>D</td>
<td>E</td>
<td>CE</td>
</tr>
<tr>
<td>E</td>
<td>BC</td>
<td>BDE</td>
<td>BCE</td>
<td>BD</td>
<td>AC</td>
<td>DE</td>
<td>C</td>
<td>ACD</td>
<td>AC</td>
</tr>
<tr>
<td>CDE</td>
<td>BCD</td>
<td>BCDE</td>
<td>CE</td>
<td>BE</td>
<td>CDE</td>
<td>AD</td>
<td>CD</td>
<td>ACD</td>
<td>AC</td>
</tr>
<tr>
<td>AE</td>
<td>AD</td>
<td>AC</td>
<td>AB</td>
<td>A</td>
<td>B</td>
<td>BCE</td>
<td>ABCD</td>
<td>ABCE</td>
<td>ABC</td>
</tr>
<tr>
<td>ACD</td>
<td>ABCD</td>
<td>ABDE</td>
<td>ACE</td>
<td>ABD</td>
<td>BCDE</td>
<td>ABCDE</td>
<td>AB</td>
<td>BD</td>
<td>BC</td>
</tr>
<tr>
<td>ACDE</td>
<td>ABCDE</td>
<td>A</td>
<td>ABE</td>
<td></td>
<td>ABDE</td>
<td>BCD</td>
<td>ABD</td>
<td>ABC</td>
<td>BE</td>
</tr>
<tr>
<td>CD</td>
<td>A</td>
<td>A</td>
<td>ABCE</td>
<td>ADE</td>
<td>ADE</td>
<td>ABC</td>
<td>BDE</td>
<td>AB</td>
<td>ABE</td>
</tr>
</tbody>
</table>

$X_1$ $X_2$ $X_3$ $X_4$ $X_5$

For the latter approach, we first assume that $\Omega_i = \psi_i \times \pi_i$, for $i = 1, 2$. Then one can extract $\psi_1$ and $\psi_2$ using the function `star_to_spread()` on $\text{starn5t3a}$ and $\text{starn5t3b}$, respectively.

R> red_spread_1 <- star_to_spread(starn5t3a)
R> red_spread_2 <- star_to_spread(starn5t3b)
R> psi_1 <- vector_to_string(red_spread_1[[1]])
R> psi_1

[1,] "A"  "B"  "C"  "D"  "AB"
[2,] "ABC" "CD" "ABD" "ACD" "BD"
[3,] "BC" "BCD" "ABCD" "AC" "AD"

R> psi_2 <- vector_to_string(red_spread_2[[1]])
R> psi_2

[1,] "C"  "AD" "B"  "A"  "D"
[2,] "ABCD" "AB" "CD" "BC" "AC"
[3,] "ABD" "BD" "BCD" "ABC" "ACD"

The output of `star_to_spread()` is a list containing two values, the reduced spread and a collineation matrix over $\mathcal{P}_5$ that ensures that the spreads $\psi_i$ spans $\langle A, B, C, D \rangle$ and the nucleus is $\pi_i = \langle E \rangle$. For instance, the collineation matrix for relabelling $\Omega_2$ is obtained as

R> red_spread_2$collineation

$X_1$ $X_2$ $X_3$ $X_4$ $X_5$

0 0 0 0 1
0 0 0 1 0
and the corresponding relabelled star can be extracted as follows.

```r
R> Cstarn5t3b <- applyCollineation(red_spread_2$collineation,starn5t3b)
R> vectortostring(Cstarn5t3b)
[1,]  "E"  "AD"  "B"  "A"  "ACDE"
[2,]  "CE"  "AB"  "CDE"  "ABCE"  "D"
[3,]  "ABCDE"  "BD"  "BCDE"  "BCE"  "ACE"
[4,]  "C"  "ADE"  "BE"  "AE"  "E"
[5,]  "ABDE"  "ABE"  "CD"  "BC"  "DE"
[6,]  "ABCD"  "BDE"  "BCD"  "E"  "AC"
[7,]  "ABD"  "E"  "E"  "ABC"  "ACD"
```

We now check the isomorphism between $\Omega_1$ and $\Omega_2$ by simply checking the isomorphism between `red_spread1[[1]]` and `red_spread2[[1]]`.

```r
R> checkSpreadIsomorphism(red_spread1[[1]], red_spread2[[1]],
+   returnfirstIEC=TRUE)

$result
[1] TRUE

$IECs[[1]]
[1,] 1 1 1 1
[2,] 0 0 1 1
[3,] 1 1 1 0
```

Example 5 Bingham et al. (2008) considered a $2^5$ factorial split-lot design for manufacturing plutonium alloy cookies which involved five basic factors with three processing stages. They suggested a design $PA_1 = \{\langle A, B, CDE \rangle, \langle C, AD, BE \rangle, \langle D, E, ABC \rangle\}$. This design is derived from a balanced covering star $St(5, 5, 3, 1)$ of $P_5$. Later on, Ranjan et al. (2010) recommended another design $PA_2 = \{\langle A, B, DE, ACD \rangle, \langle C, AB, DE, ACD \rangle, \langle D, E, AB, ACD \rangle\}$, which represents a balanced covering star $St(5, 3, 4, 3)$ of $P_5$.

We can define the two stars as shown in the following code. For defining $PA_1$, `basistoflat3()` was presented in Example 3. We similarly write a function called `basistoflat4()` for constructing the $(4-1)$-flats, and then define $PA_2$ as follows.

```r
R> n <- 5
R> flatsize <- 2^3-1
R> mu <- 3

R> star_PA1 <- array(NA, c(n, flatsize, mu))
R> star_PA1[,1] <- basistoflat3("A", "B", "CDE", n)
R> star_PA1[,2] <- basistoflat3("C", "AD", "BE", n)
R> star_PA1[,3] <- basistoflat3("D", "E", "ABC", n)
```
```r
R> n <- 5
R> flatsize <- 2^4-1
R> mu <- 3
R> star_PA2 <- array(NA, c(n, flatsize, mu))
R> star_PA2[,1] <- basistoflat4("A", "B", "DE", "ACD", n)
R> star_PA2[,2] <- basistoflat4("C", "AB", "DE", "ACD", n)
R> star_PA2[,3] <- basistoflat4("D", "E", "AB", "ACD", n)

The isomorphism between the two stars can be checked by using the following code.

R> checkStarIsomorphism(star_PA1, star_PA2)

Stars are not of same dimension.
[1] FALSE

The output is certainly expected as the flatsize in PA1 and PA2 are different. One can also verify that the sizes of nucleus in these two stars are different.

R> nucleus_PA1 <- suppressMessages(plyr::match_df(data.frame(t(star_PA1[,1])),
+ data.frame(t(star_PA1[,2]))))
R> vectortostring(as.numeric(nucleus_PA1))

[,1]
[1,] "ABCDE"

R> nucleus_PA2 <- suppressMessages(plyr::match_df(data.frame(t(star_PA2[,1])),
+ data.frame(t(star_PA2[,2]))))
R> vectortostring(t(as.matrix(nucleus_PA2)))

[,1]
[1,] "DE"
[2,] "ACD"
[3,] "AB"
[4,] "ACE"
[5,] "ABDE"
[6,] "BCD"
[7,] "BCE"
```

5. Catalog of Multi-Stage Factorial Designs

In this section, we illustrate the usage of Consistency functions in catalog construction for balanced spread- and balanced covering star- based designs. We do not claim to propose any new designs, new ranking criterion, or new catalogs. In general, the construction of such a catalog requires iterating through all possible candidate designs and then checking isomorphism among them (Bingham and Sitter, 1999b; Lin and Sitter, 2008). Unfortunately, we are unaware of any computationally efficient methodologies that can construct all possible balanced $(t - 1)$- spreads or balanced covering stars of $PG(n - 1, 2)$. As a result, we follow an exhaustive search approach for constructing all possible spreads.

Researchers and practitioners often combine such catalogs with a carefully chosen design ranking criteria to sort through different designs and select the most optimal candidate for their experiments.
For classical fractional factorial designs, popular design ranking criteria include maximum resolution, generalized minimum aberration, maximum number of clear two-factor interactions (2-fi’s), etc. (Xu et al., 2001; Butler, 2003; Cheng and Tsai, 2011). Let $d$ be a factorial design with randomization restrictions characterized by $S_1, ..., S_k$, then, Bingham et al. (2008) proposed a $V$-criterion defined by

$$V(d) = \sum_{i=1}^{k} (p_i - \bar{p})^2,$$

where $\bar{p} = \sum_{i=1}^{k} p_i / k$ and $p_i$ is proportion of main effects and 2-fi’s in $S_i$. If we have a $(t - 1)$-spread based design then $S_i$ would be an RDCSS, or equivalently, a $(t - 1)$-flat in the $(t - 1)$-spread, whereas, if the design is obtained from a star $St(n, \mu, t, t_0)$, then $S_i$ would be the flat minus the nucleus (i.e., $|S_i| = 2^t - 2^{t_0}$). From an analysis standpoint, if we use half-normal plots for assessing the significance of factorial effects, then $S_i$ represents the set of factorial effects that would be used for the $i$-th half-normal plot. Thus minimizing this $V$-criterion encourages homogeneous distribution of main effects and two-factor interactions among the $S_i$’s.

We now discuss building a catalog of designs characterized by a $(t - 1)$-spread of $P_n$, for fixed $t < n$. Suppose we wish to build a catalog of 1-spreads of $P_n = PG(3, 2)$ (Table 3 presents an example spread1t2a). This structure contains $2^4 - 1 = 15$ elements (factorial effects) distributed in 5 subspaces of size 3 each. One possible (although computationally inefficient) approach is to exhaustively search all possible permutations/arrangements of these 15 elements, and then check whether or not they form a 1-spread. If yes, then we can check the isomorphism between this spread and the original spread1t2a. We can also compute the design ranking criterion value for all these spreads. The steps of catalog construction are summarized in Algorithm 3.

**Algorithm 3** for constructing a catalog of non-isomorphic spreads (or stars).

1. Pick an initial $(t - 1)$-spread (say, $d_1$) of $P_n := PG(n - 1, 2)$.
2. Compute the design ranking criterion value of $d_1$.
3. Pick a random permutation (say $\rho$) of the set of $2^n - 1$ elements of $d_1$.
4. Use is.spread() to check if $\rho(d_1)$ is a proper $(t - 1)$-spread of $P_n$. If “TRUE”, go to the next step, otherwise go back to Step 3 (if all possible permutations of $d_1$ have not been exhausted yet).
5. Compute the design ranking criterion value of $\rho(d_1)$.
6. Use checkSpreadIsomorphism() to check whether or not $\rho(d_1)$ is isomorphic to $d_1$.
7. Go to Step 3 if all possible permutations of $d_1$ have not been exhausted yet.

Note that $15! = 1.307674 \times 10^{12}$ is a big number and one would have to iterate through these many possible rearrangements to ensure the construction of all possible 1-spreads of $P_4$. Admittedly, one can come up with innovative ideas to save some computational cost, but that is not the main objective here. For this example, we ran this algorithm for over 5 million random permutations, and found almost 110 isomorphic spreads. Note that we did not expect to find any non-isomorphic spreads, because Spencer et al. (2019) have already mentioned that all 1-spreads of $PG(3, 2)$ are isomorphic. Nonetheless, we used this spread for illustration because it is a computationally inexpensive small example to follow through. Interestingly, we found designs with two criterion value $V(d) = 0.44$ and 0.22, which indicates preference of one design over the other. The initial design spread1t2a gave $V(d) = 0.22$ (with $p_1 = 2/3, p_2 = 2/3, p_3 = 1/3, p_4 = 2/3$ and $p_5 = 1$), whereas, the following design, found through Algorithm 3, corresponds to $V(d) = 0.44$ (with $p_1 = 1/3, p_2 = 2/3, p_3 = 1/3, p_4 = 1/3$ and $p_5 = 1$).

```plaintext
[1,1] "ABC" "CD" "AD" "ABD" "B"
```
Since the objective is to find designs with lower $V(d)$ criterion value, although isomorphic, designs with $V(d) = 0.22$ is better than the one with $V(d) = 0.44$.

If the underlying design is based on a star (say $\Omega$), then one can use `star_to_spread(\Omega)` function to extract the spread $\psi$ after removing the nucleus $\pi$ (assuming $\Omega = \psi \times \pi$) from the star. Subsequently, the designs from the catalog of the underlying spreads can be combined with the nucleus to construct the desired catalog of stars. For instance, `star_to_spread(starn5t3a)` gives a 1-spread of $PG(3, 2)$ (discussed in details).

6. Concluding Remarks

This paper presents an R package `IsoCheck` to check the isomorphism of balanced $(t - 1)$-spreads and balanced covering stars of $\mathcal{P}_n = PG(n - 1, 2)$, which corresponds to the isomorphism check of two-level multi-stage factorial designs with randomization restrictions (Ranjan et al., 2009, 2010). The main ideas behind the algorithms are to apply the relabellings via collineation matrices and rearrangements or the factorial effects within the RDCSSs through bitstring representation of the elements of $\mathcal{P}_n$. In this paper, we have presented several worked out examples which illustrate the usage of `IsoCheck` functions and their arguments for the isomorphism check. More importantly, we also outline how `IsoCheck` functions can be used to build catalog of good designs in this class of spread/star-based factorial designs.

A few important remarks are in order. 1. In practice, whenever we have more than one possible design options for conducting an experiment, the ranking of design becomes important. This has traditionally been done in the context of “fractional” factorial designs, as we have several options for choosing the fraction defining contrast subgroups (FDCS). For multi-stage factorial designs with randomization restrictions (i.e., spread/star -based designs), we have to choose even randomization defining contrast subspaces (RDCSSs), which are in some sense similar to FDCS. Therefore, we can also talk about the isomorphism and ranking of the full factorial unreplicated spread/star-based multi-stage factorial designs with randomization restrictions. 2. One can use `IsoCheck` functions to check for the isomorphism of multi-stage fractional factorial designs as well. 3. The generalization of these isomorphism check algorithms for $q(\geq 3)$-level factorial designs appears to be non-trivial.

Acknowledgement

The authors would like to thank the Editor and the reviewers for their helpful comments which led to substantial improvement in the manuscript.

REFERENCES


910  ISOCHECK: R PACKAGE FOR ISOMORPHISM CHECK


