Behavior Research Methods (2020) 52:1459–1468
https://doi.org/10.3758/s13428-019-01326-x

An algorithm to minimize the number of blocks in incomplete block designs

Justin A. MacDonald[1] · Michael C. Hout[1] · Joseph Schmidt[2]

Published online: 10 December 2019

Abstract
Incomplete block designs are experimental designs in which a subset of treatments are included in each block. The researcher must decide which conditions are assigned to each block. This design concept is quite general. At the level of the experiment, a treatment is a condition in an experiment, blocks are different groups of subjects, and the researcher must decide how to assign a subset of conditions to each block of subjects. At the level of the subject, the treatments correspond to individual stimuli, blocks correspond to experimental trials, and the researcher must decide which subset of stimuli to include in each trial. In this article, we present an efficient algorithm that assigns treatments to blocks in an incomplete block design according to two criteria: Each pair of treatments must appear together in at least one block, and the number of blocks in the experiment is minimized. We discuss details and applications of the algorithm and provide software and a web application to generate designs according to the needs of the researcher.

Keywords Experiment design · Incomplete block dezsigns · Randomized block designs · Set cover problem

A *randomized complete block design* is an experiment in which all treatments are tested in every block (Hays, 1994).[1] If there are$n$ treatments and b blocks, then there are $r = b$ replications per treatment. In addition, there are $\binom{n}{2}$ different pairs of treatments,[2] and each pair of treatments occurs together exactly once within each block. Therefore, there are $\lambda = b$ observations of each pair. In contrast, an *incomplete block design* is an experiment in which a subset of treatments are tested during each block. If each block includes k treatments ($k < n$), then the number of replications r can vary across treatments, and the number of observations of each treatment pair (λ) can vary across pairs, as well. This leads to collecting unequal amounts of data across treatments, which will cause the precision of the resulting parameter estimates to vary. This undesirable effect can be counteracted by constructing a *balanced incomplete block design* (BIBD), in which (1) each block contains k treatments, where $k < n$; (2) all n treatments occur exactly r times; and (3) all $\binom{n}{2}$ pairs of treatments occur exactly λ times (Bose, 1949; Kimmel, Becker, & Bevill, 1988). Unfortunately, BIBDs exist for only a small subset of the possible combinations of the parameters n, k, b, r, and λ (Clay, 1972; Cochran & Cox, 1957; Fisher & Yates, 1963). The two necessary but not sufficient conditions for the existence of a BIBD (when $\lambda = 1$) are that $n(n-1)$ must be an integer multiple of $k(k-1)$, and $(n-1)$ must be an integer multiple of $(k-1)$ (Hanani, 1975). Figure 1 illustrates the combinations of $3 \le n \le 200$ and $2 \le k \le 50$ that meet these necessary conditions. Coverage is quite sparse, and even when the necessary conditions are met, the existence of a BIBD is not guaranteed.

In the interest of covering the entire parameter space, therefore, this article is *not* focused on the construction of BIBDs. Instead, we propose constructing block designs according to slightly different criteria that will allow for coverage of the entire parameter space. From a pragmatic point of view, the experimenter almost always wants to minimize the length of the experiment (the number of blocks, b). At the same time, the experimenter may wish to be certain that each pair of

[1] Please note that we are using the term "treatment" quite generally here. "Item" or "stimulus" could be substituted instead. Similarly, "trial" could be substituted for "block."

[2] In general, $\binom{n}{k}$ refers to the number of combinations of n objects taken k at a time and is equal to $\frac{n!}{k!(n-k)!}$.

✉ Justin A. MacDonald
jmacd@nmsu.edu

[1] New Mexico State University, Las Cruces, NM, USA

[2] University of Central Florida, Orlando, FL, USA

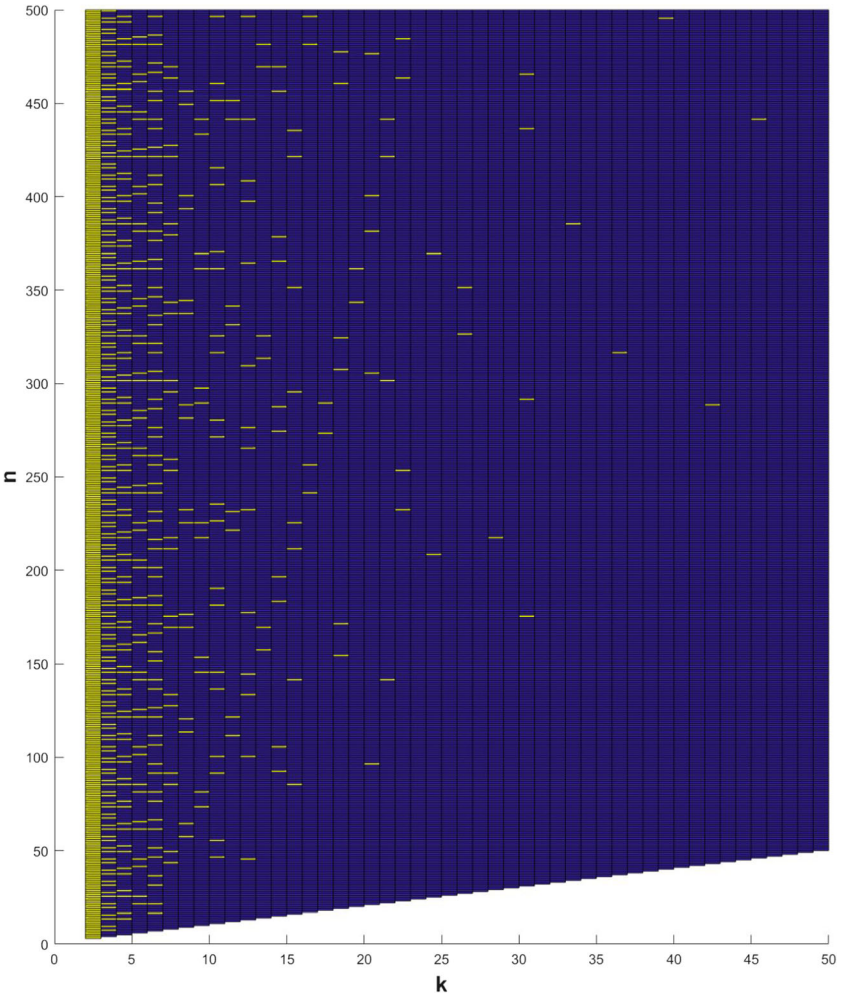

Fig. 1 Combinations of n and k that meet the necessary conditions for the existence of a balanced incomplete block design (BIBD). Yellow marks indicate that the conditions are met, and blue indicates that they are not met. Note that in general, these conditions are not sufficient, so the existence of a BIBD is not certain even when the necessary conditions are met.

treatments appears together in a block a minimum number of times (denoted as λ_0). This criterion is appropriate when designing experiments in which stimuli must be presented in a controlled fashion, such as when comparing the similarity of items for use in multidimensional scaling (Berman et al., 2014; Horst & Hout, 2015; Hout, Cunningham, Robbins, & MacDonald, 2018; Hout et al., 2015; Hout, Papesh, & Goldinger, 2012), when balancing stimuli over screen locations, or when a set of stimuli are varied in systematic fashion so as to create variable contexts, as might occur when selecting a larger set of "foils" for a memory test. Therefore, our design problem is as follows: for given values of n and k, if we denote the number of times treatments i and j appear together as λ_{ij}, then we want to minimize b subject to the constraint that $\lambda_{ij} \geq \lambda_0$ for all values of i and j such that $i \neq j$.

A trivial way to reduce the number of blocks (b) is to increase the size of each block (k): more treatments per block allows for a greater number of treatments to appear together within a block, thereby reducing the number of blocks necessary. However, the number of treatments per block is often constrained by practical considerations, rendering an incomplete block design necessary. For example, a researcher could be planning an experiment in which each treatment is quite time-consuming and the inclusion of all treatments within a single block is impractical. As another example, we often run studies that utilize the spatial-arrangement method (Goldstone, 1994; Hout, Goldinger, & Ferguson, 2013; Hout & Goldinger, 2016) to collect pairwise similarity judgments for many different visual stimuli (e.g., Hout, Goldinger, & Brady, 2014). This procedure is illustrated in Fig. 2. The method involves presenting multiple images (i.e., treatments) on a computer screen and asking the subject to manipulate the image positions to indicate the images' visual similarity. Ideally, we would present all n images on the screen during a single trial, and all pairwise similarity judgments could be completed at once (this would represent a randomized block design). However, most often there are far too many stimuli to allow for presenting them all on the screen at the same time. We

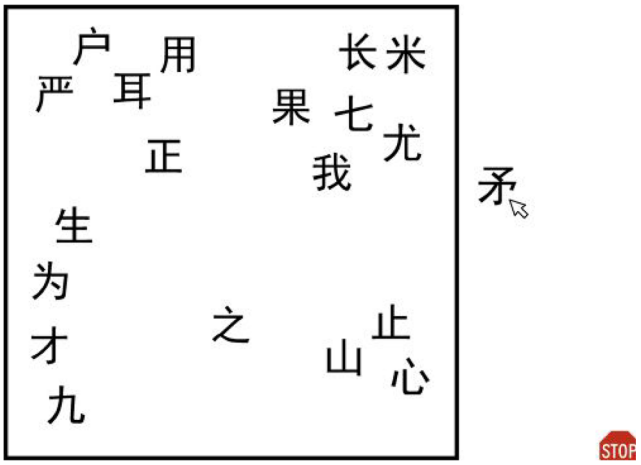

Fig. 2 The spatial-arrangement method. In each trial, the images start outside the central "arena"; they are then moved in and arranged such that the distance between each pair of objects represents the user's perceived similarity of the items. Once all of the objects have been arranged using the mouse, the participant clicks the Stop sign to terminate the trial. This procedure often requires an incomplete block design, because there are typically many more stimulus pairs to be tested than can feasibly be displayed on the computer screen during a single trial.

therefore must choose a subset of k images (where $k \ll n$) to present during each trial. This represents an incomplete block design, as similarity judgments are collected on only a subset of the $\binom{n}{2}$ image pairs during each trial. Accordingly, we want to design an incomplete block experiment in which the similarity of each image pair is judged at least λ_0 times over the course of the experiment while keeping the number of trials (b) to a minimum. To minimize the length of the experiment, it is necessary include as many stimuli on the screen as is practical (in other words, to increase k as much as is practical) and design the experiment so that the number of trials (b) is minimized. Of course, in the situations in which the number of treatments per block is not constrained, one can just use a complete block design.

The purpose of this article is to present an algorithm that constructs block designs with given values for n and k, according to the criteria outlined above. For simplicity we will focus on designs for which λ_0 is equal to 1.[3] As a start, consider a standard greedy algorithm that builds a design block-by-block, searching through the complete set of blocks to find the one that includes the maximal number of still-to-be-included treatment pairs. This approach can be formalized as follows:

1. Out of all $\binom{n}{k}$ possible blocks, choose the one that includes the most uncovered (still-to-be-included) pairs.
2. Reduce the list of uncovered pairs appropriately.
3. Repeat Steps 1 and 2 until all pairs are covered.

This straightforward approach will lead to good (although not necessarily optimal) solutions to the problem (Vazirani, 2003). However, the number of possible blocks gets prohibitively large with increasing n and k. In an experiment with $n = 50$ treatments and $k = 4$ treatments per block, for example, there are 230,300 possible blocks, each of which would be evaluated many times while building a design for the experiment. If the experimenter wanted to include 20 treatments in a given block instead of four (a realistic choice for a typical visual similarity judgment task we might conduct in our labs, for example), the number of blocks to be considered is more than 47 trillion, rendering the approach prohibitively time-consuming. Clearly, this approach must be revised to allow for practical use with moderate or large values of n and k. That is precisely what we accomplish here.

Proposed algorithms

Importantly, a successful algorithm should have three properties. First, it should be flexible; that is, it should produce solutions for any combination of n and k such that $n \geq k \geq 2$. Second, it should be less computationally intensive than a standard greedy algorithm, so that it can be employed with a large number of treatments. Third, it should be efficient; that is, it should produce designs that minimize b as much as possible, so as to minimize participant investment/workload.

The most straightforward approach is to choose treatments randomly for each block and then continuously evaluate coverage until all $\binom{n}{2}$ treatment pairs have been covered.

1. To generate a block, choose k treatments at random (without replacement) from the set of n treatments.
2. Repeat Step 1 until all treatment pairs are covered.

We will refer to this approach as the *Random* strategy. Given that treatment selection here is entirely non-strategic, we consider this the worst possible algorithm that will still produce solutions to the problem. Alternatively, a more deliberate (and computationally intensive) approach is to generate many random blocks, and then choose the block from the sample that covers the most still-to-be-included pairs:

1. To generate a block, create m candidate blocks, each by choosing k treatments at random without replacement from the set of n treatments.

[1] This design problem is a special case of the Steiner system problem (Hanani, 1975). A Steiner system is a block design in which there are n treatments, each block contains k treatments ($k < n$), and each pair of treatments occurs together in a block exactly once. Thus, if we set $\lambda_0 = 1$, our design problem is equivalent to finding an approximate $S(2, k, n)$ Steiner system.

2. Choose the candidate block that covers the most uncovered pairs.
3. Repeat Steps 1 and 2 until all pairs are covered.

In this article we set m to 2,000, which in our view represents a reasonable trade-off between computational complexity and the quality of the solutions produced by the algorithm. We will refer to this as the *Best of Random* algorithm. This can be considered a middle-of-the-road approach: That is, it is nowhere near optimal, but it is certainly an algorithm that *could* produce reasonable solutions. Finally, we will test two strategies related to the greedy algorithm discussed previously. The greedy algorithm requires searching through all $\binom{n}{k}$ possible blocks to pick the one that includes the most uncovered treatment pairs. Our modified greedy algorithm will instead build an experimental design by making greedy choices at the level of the individual treatment, rather than at the level of the block. Instead of searching through $\binom{n}{k}$ blocks to find the best block, our modified greedy algorithm will search through n treatments to choose the one that, if included in the block, would result in the most treatment pairs being covered. For simplicity, we will refer to this as the *Greedy* algorithm. It chooses individual treatments to include in a block as follows:

1. Identify the treatment that, if included in the current block, would result in the coverage of as many uncovered pairs as possible. If more than one treatment is associated with this maximum, proceed to Step 2.
2. From the finalists identified in Step 1, choose the treatment that is included in the most uncovered pairs. If more than one treatment is associated with this maximum, proceed to Step 3.
3. From the finalists in Step 2, select one treatment at random.

This process repeats until all pairs have been covered. To illustrate the process with a very simple example, let $n = 3$ and $k = 2$. In this case, $\binom{3}{2} = 3$ pairs are to be covered, and each block will consist of two treatments. The algorithm proceeds selecting treatments one at a time until all three pairs are covered: (1, 2), (1, 3), and (2, 3).

Block 1, Treatment 1

1. No treatments are already chosen for the block, so including any one of the treatments will not cover any uncovered pairs. Therefore, all three treatments are tied, and we proceed to Step 2.
2. All three treatments are finalists from Step 1. From this set of finalists, choose the treatment that is included in the most uncovered pairs. Each finalist is included in two uncovered pairs, so we proceed to Step 3.
3. All three treatments are finalists from Step 2. Treatment 3 is chosen randomly from this group.

Pairs covered: none.

Block 1, Treatment 2

1. Treatment 3 has already been chosen for the current block. Including Treatments 1 or 2 would each result in one pair being covered: if Treatment 1 were included, the pair (1, 3) would be covered, and if Treatment 2 were included, the pair (2, 3) would be covered. Therefore, Treatments 1 and 2 are tied, and we proceed to Step 2.
2. Treatments 1 and 2 are finalists from Step 1. From this set of finalists, choose the treatment that is included in the most uncovered pairs. Each finalist is included in two yet-to-be-covered pairs, so we proceed to Step 3.
3. Treatments 1 and 2 are finalists from Step 2. Treatment 1 is chosen randomly from this group.

Pairs covered: (1, 3).

Block 2, Treatment 1

1. No treatments are already chosen for the block, so including any one of the treatments will not cover any yet-to-be covered pairs. Therefore, all three treatments are tied, and we proceed to Step 2.
2. All three treatments are finalists from Step 1. From this set of finalists, choose the treatment that is included in the most yet-to-be-covered pairs. Treatment 1 is included in the uncovered pair (1, 2), Treatment 2 is included in (1, 2) and (2, 3), and Treatment 3 is included in (2, 3). Therefore, Treatment 2 is chosen.

Pairs covered: (1, 3).

Block 2, Treatment 2

1. Treatment 2 has already been chosen for the current block. Including Treatments 1 or 3 would each result in one pair being covered: If Treatment 1 were included, the pair (1, 2) would be covered, and if Treatment 3 were included,

the pair (2, 3) would be covered. Therefore, Treatments 1 and 3 are tied, and we proceed to Step 2.
2. Treatments 1 and 3 are finalists from Step 1. From this set of finalists, choose the treatment that is included in the most uncovered pairs. Each finalist is included in one uncovered pair, so we proceed to Step 3.
3. Treatments 1 and 3 are finalists from Step 2. Treatment 1 is chosen randomly from this group.

Pairs covered: (1, 2), (1, 3).

Block 3, Treatment 1

1. No treatments are already chosen for the block, so including any one of the treatments will not cover any uncovered pairs. Therefore, all three treatments are tied, and we proceed to Step 2.
2. All three treatments are finalists from Step 1. From this set of finalists, choose the treatment that is included in the most uncovered pairs. Treatment 1 is not included in any uncovered pairs, Treatment 2 is included in (2, 3), and Treatment 3 is included in (2, 3). Therefore, Treatments 2 and 3 are tied, and we proceed to Step 3.
3. Treatments 2 and 3 are finalists from Step 2. Treatment 2 is chosen randomly from this group.

Pairs covered: (1, 2), (1, 3).

Block 3, Treatment 2

1. Treatment 2 has already been chosen for the current block. Including Treatment 1 would not cover any outstanding pairs, and including Treatment 3 would result in (2, 3) being covered. Therefore, Treatment 3 is chosen.

Pairs covered: (1, 2), (1, 3), (2, 3).

Because all pairs are now covered, the process terminates, and the block list (i.e., the experiment design) is output: {(3, 1), (2, 1), (2, 3)}. In this simple situation, it is obvious that the algorithm produced the optimal solution: There are three pairs to be covered, and only one pair can be covered in each block. Therefore, three blocks are required to cover all pairs.

Finally, we will test a version of the Greedy algorithm in which the above process is run m times, and the design with the fewest blocks is chosen. This is potentially useful because the Greedy algorithm is nondeterministic, and if run multiple times, it will produce designs that vary somewhat in the number of blocks. We chose to set $m = 100$ for this comparison, and will refer to this as the *Best of Greedy* algorithm.

To summarize the four approaches considered in this article:

- *Random*: Choose treatments randomly for each block. Continue until all treatment pairs have been covered.
- *Best of Random*: To determine the treatments in a block, create 2,000 candidate blocks at random. Out of the 2,000 blocks, choose the one that covers the most still-to-be-covered treatment pairs. Continue until all treatment pairs have been covered.
- *Greedy*: Assign treatments to blocks according to our greedy algorithm, as outlined above.
- *Best of Greedy*: Run our Greedy algorithm 100 times, which will produce 100 experiment designs. Choose the design that has the smallest number of blocks.

We expected the Random, Best of Random, Greedy, and Best of Greedy algorithms to vary both in the size of the designs they produced and in the time taken to produce them. Because computation time is platform- and implementation-dependent, however, we could only consider computation time in a noncomparative sense, providing the reader with a general expectation of the time required to run the algorithms for various combinations of n and k. Our main performance metric was therefore the size of the solutions produced (i.e., the number of blocks in the provided design). We expected Random to provide the worst (i.e., largest) solutions of the four approaches. Best of Random should provide better solutions, as it selects the best-performing randomly constructed blocks. We expected the Greedy algorithm to outperform Random and Best of Random, due to the strategic (nonrandom) way in which blocks are constructed. Finally, we expected Best of Greedy to outperform all the other algorithms. These expectations were tested using a series of simulations conducted in Matlab (The Mathworks, Natick, MA). The simulations were conducted for all combinations of $n = \{4, 5, \ldots, 200\}$ and $k = \{3, 4, \ldots, 50\}$ for which $n > k$. The best-performing algorithm was subjected to additional testing, as well.

Results

Number of blocks

The overall performance for the four algorithms is illustrated in Figs. 3 and 4. As expected, Random consistently produced the worst solutions, exceeding the other algorithms by as much as 28,000 blocks. In fact, the Random algorithm can be eliminated from contention outright, at least in terms of the length of the solutions it provides. Out of the 8,328 tested combinations of n and k, Random never outperformed the other algorithms. It produced the best (smallest) solution 189 times (2.2% of tested combinations), but in all of these instances, performance was also matched by one of the other algorithms. The other three algorithms were slightly more

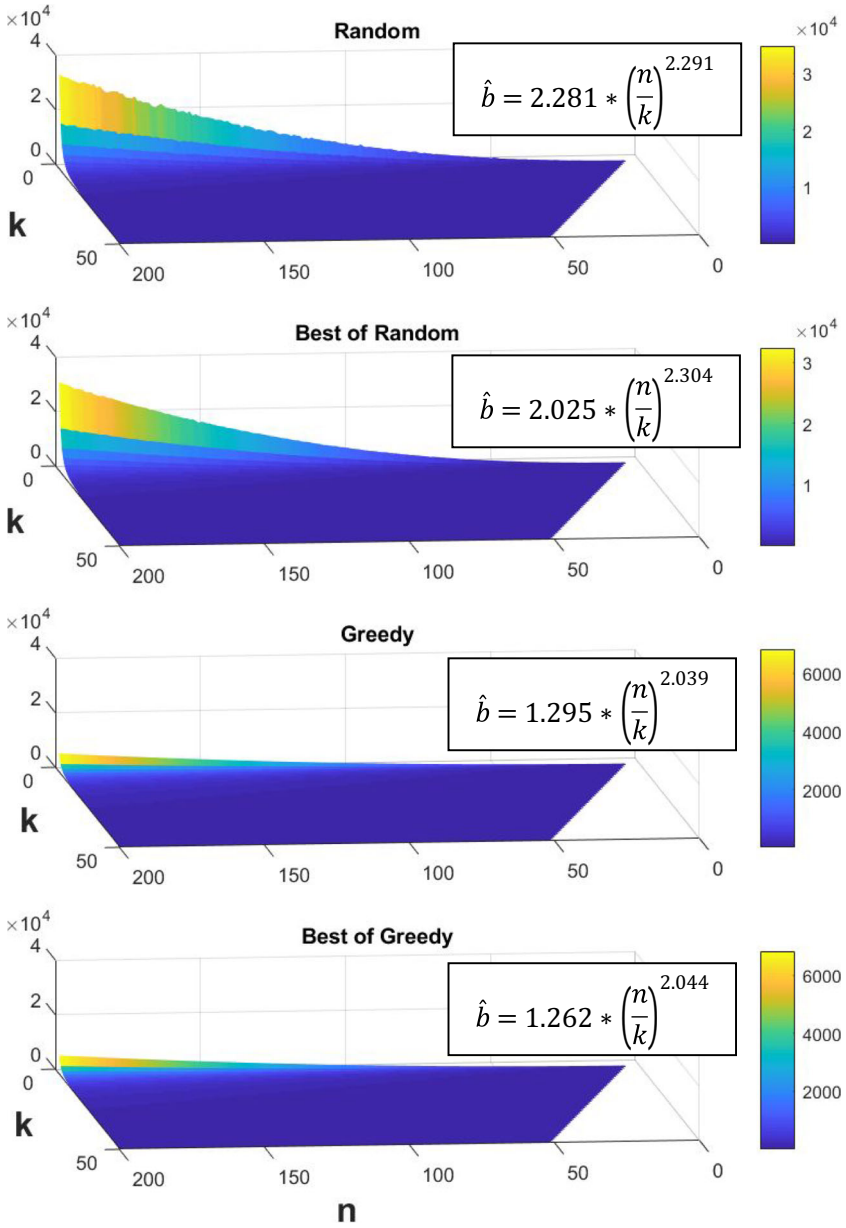

Fig. 3 Overall performance of the selection algorithms. The height of the surface indicates the number of blocks in the solution. The largest solutions were required for large n and small k. Please note that the color scales differ across the surface plots. Power functions were fit to each surface, with $^{n}/_{k}$ as the predictor. Each of the fits accounted for more than 99% of the variance in number of blocks.

competitive. Best of Random produced the outright best solution 40 times (0.5%) and tied with Greedy or Best of Greedy for the best solution 731 times (8.8%). Greedy produced the outright best solution 20 times (0.2%) and tied with Best of Random or Best of Greedy 2,363 times (28.4%). Finally, Best of Greedy produced the outright best solution 5,904 times (70.9%), and tied with Best of Random or Greedy 2,364 times (28.4%). In summary, the Random algorithm produced the best solution in 2.2% of the tested combinations of n and k, but in all of these instances one of the other algorithms produced an equally good solution. Predictably, Best of Random performed better, producing the best solution for 9.3% of the tested combinations. The Greedy and Best of Greedy algorithms performed better yet, producing the best solutions in 28.6% and 99.3% of tested combinations, respectively. Quite clearly, the Best of Greedy approach is superior.

Computation time

For all algorithms the largest computation times were for solutions that require many blocks, typically for combinations of large n and small k. Random algorithm solution times ranged

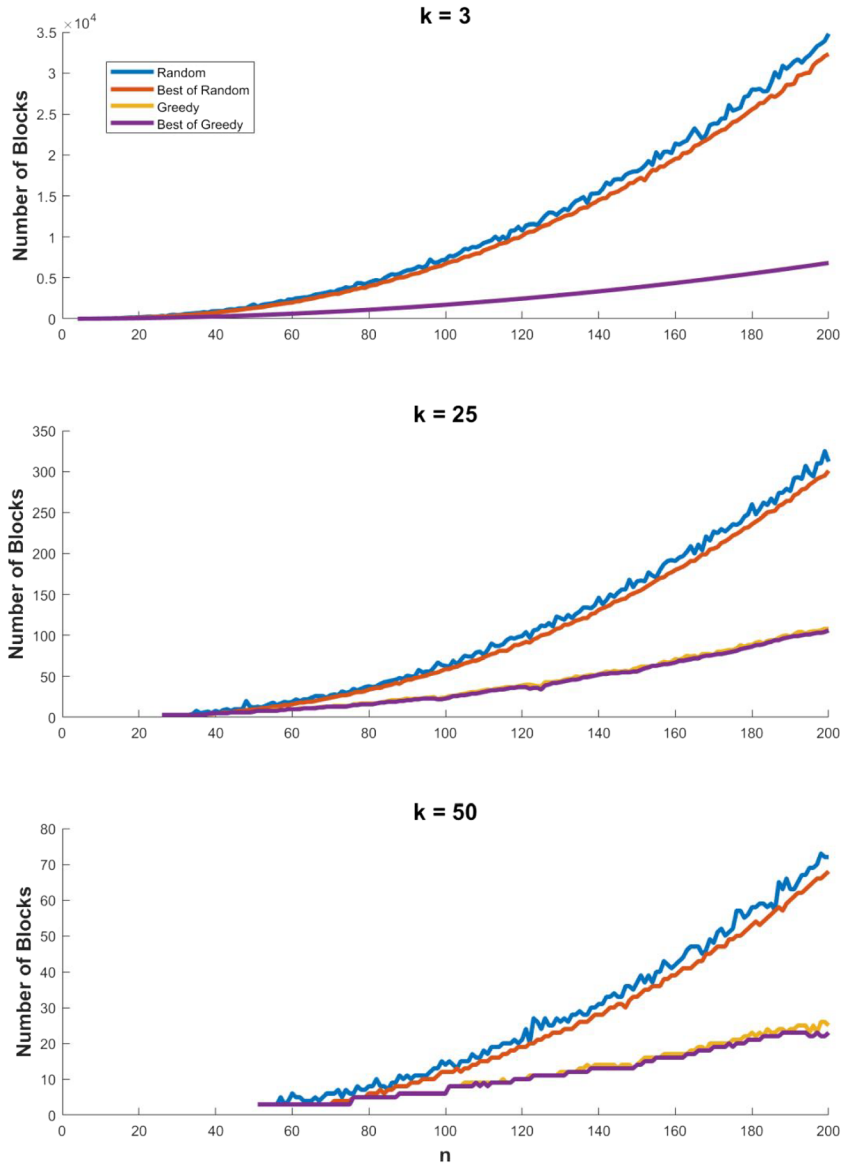

Fig. 4 Algorithm performance for $k = 3$, 25, and 50. The Greedy and Best of Greedy algorithms matched or outperformed their Random counterparts for all values of n. For $k = 3$, the solution lengths provided by Greedy and Best of Greedy were the same across all values of n. The Best of Greedy and Best of Random algorithms provided an incremental improvement over Greedy and Random, respectively.

from 0.001 to 2.457 s (mean: 0.037 s, median: 0.004 s). Best of Random solution times ranged from 0.047 to 1000 s (mean: 14.83 s, median: 1.488 s). Greedy solution times ranged from 0.001 to 1.639 s (mean: 0.089 s, median: 0.046 s). Finally, Best of Greedy solution times ranged from 0.039 to 36.155 s (mean: 1.732 s, median: 0.888 s). These results were produced on a computer that was moderately powerful at the time that this article was written. Of course, algorithm computation times are both platform- and implementation-dependent, and therefore a detailed analysis is not particularly informative. However, these results demonstrate that all four algorithms produce experiment designs in very little time, making them practical choices for the generation of designs for psychology experiments.

Concurrence matrices

Given the clear dominance of the Best of Greedy (BoG) algorithm, we decided to subject it to further testing. Another way to evaluate the algorithm is to examine the *concurrence matrices* associated with the designs that it produces, which indicate how often each pair of treatments appeared together in a block. Using our notation, the concurrence matrix contains the λ_{ij} values for a design in which $1 \leq i, j \leq n$. Because we are focusing on designs in which $\lambda_{ij} \geq \lambda_0 = 1$, the λ_{ij} values should be as close to 1 as possible in order to minimize the number of blocks for given values of n and k. Therefore, a reasonable performance metric is the mean of the λ_{ij} values, where $i \neq j$. Mean values close to 1 indicate that the design is

approaching a BIBD, and a mean value of exactly 1 indicate that the design *is* a BIBD. Mean values that deviate relatively far from 1 indicate that the design may have room for improvement. The mean λ_{ij} values for all tested combinations of n and k are shown in Fig. 5. In general, the mean values are as expected, with values close to 1 for small k, and values farther from 1 when k is close to n. Small values of k allow for less "wasted" space (repeated concurrences) during the end of the experiment, when the last few pairs are being covered. In contrast, values of k just less than n will force more repeated concurrences.

These concurrence values are arguably difficult to interpret, as only when the value is 1 can you be certain that the design is optimal (a BIBD). To this end, we examined how often the BoG algorithm produced a BIBD when the necessary conditions for a BIBD were met (Hanani, 1975). Of the 8,328 combinations of n and k that we tested, 186 meet the necessary conditions. Of course, these conditions are not sufficient, so there was no guarantee that BIBDs existed for these combinations. Out of the 186 combinations for which a BIBD *might* exist, the BoG algorithm actually found a BIBD for only two of them, indicating that the BoG algorithm is unlikely to find BIBDs in this parameter range.[4]

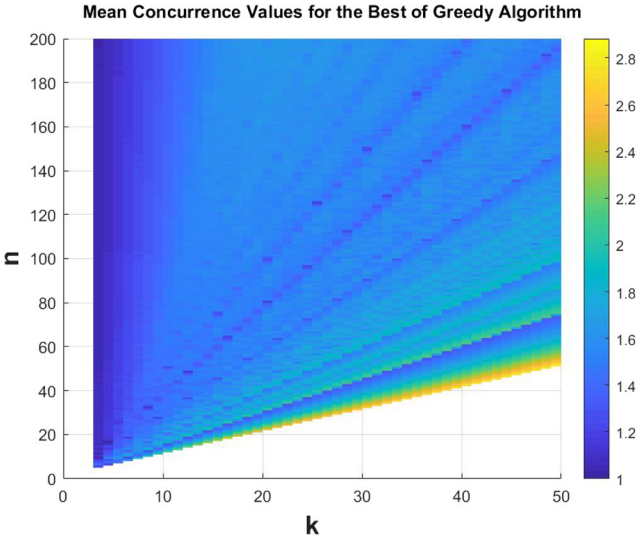

Fig. 5 Mean concurrence values for the Best of Greedy algorithm. Values near 1 indicate that the design is approaching a balanced incomplete block design.

Comparison to Mandal, Gupta, and Parsad (2014)

We also compared the BoG algorithm to that of Mandal, Gupta, and Parsad (2014). Their algorithm is widely available for use, as it is the basis of the `ibd` package in R (Mandal, 2019). By default, their algorithm requires that the user specify the number of treatments (n), the number of treatments in each block (k), and the number of blocks in the design. The algorithm will then produce a design given those parameters. The authors recommend using it for $n \leq 30$ and $k \leq 10$. Their approach is slightly different from ours, in that our algorithm minimizes the number of blocks for given values of n and k. Because the Mandal et al. algorithm requires specifying the number of blocks in advance, we adopted the following procedure for this comparison:

1. Start by setting the number of blocks by rounding $\frac{n(n-1)}{k(k-1)}$ up to the nearest integer. This number would be the size of a BIBD for given values of n and k if a BIBD were to exist. This is the smallest possible design size that could cover all treatment pairs.
2. Produce a design using the specified values for n, k, and the number of blocks.
3. Test to see whether the design covers all treatment pairs. As above, if we denote the number of times that treatments i and j appear together as λ_{ij}, then we will determine whether $\lambda_{ij} \geq \lambda_0$ for all values of i and j such that $i \neq j$.
4. If all pairs are covered, the algorithm has produced an appropriate design, and the process terminates. Otherwise, increment the number of blocks by 1, and repeat Steps 2–4.

We implemented this procedure in R (R Core Team, 2013) to produce designs for $4 \leq n \leq 19$ and $3 \leq k \leq 10$. We attempted to produce designs for values of n up to 30 (consistent with the recommendations of the algorithm authors). Unfortunately, the execution times for values of n greater than 19 were prohibitively long (spanning multiple days to produce designs for $n = 20$, for example), rendering the testing of these values impractical.

The results are illustrated in Fig. 6. The top panel illustrates the size of the Best of Greedy designs relative to those produced by the Mandal et al. (2014) algorithm. The Best of Greedy algorithm matched or outperformed the Mandal et al. algorithm across the board, with the exception of several combinations of n and small k. The difference was especially pronounced for combinations of large n and large k, with the Best of Greedy algorithm producing designs that were 30%–50% of the size of the corresponding Mandal et al. designs. The comparison of computation times reinforced again the strength of the BoG algorithm: The Mandal et al. algorithm was at least ten times slower than BoG for all combinations of n and k. For combinations of large n and k, the Mandal et al. algorithm was up to five orders of magnitude slower than Best of Greedy. In the worst case (for n = 19 and k = 8), the completion times were 74 ms for BoG and 2.3 *hours* for the Mandal et al. algorithm, and the BoG design was less than

[2] Although the performance of the BoG algorithm when $k = 2$ is not presented here, it is worth noting that the algorithm found BIBDs for all tested values of n when $k = 2$.

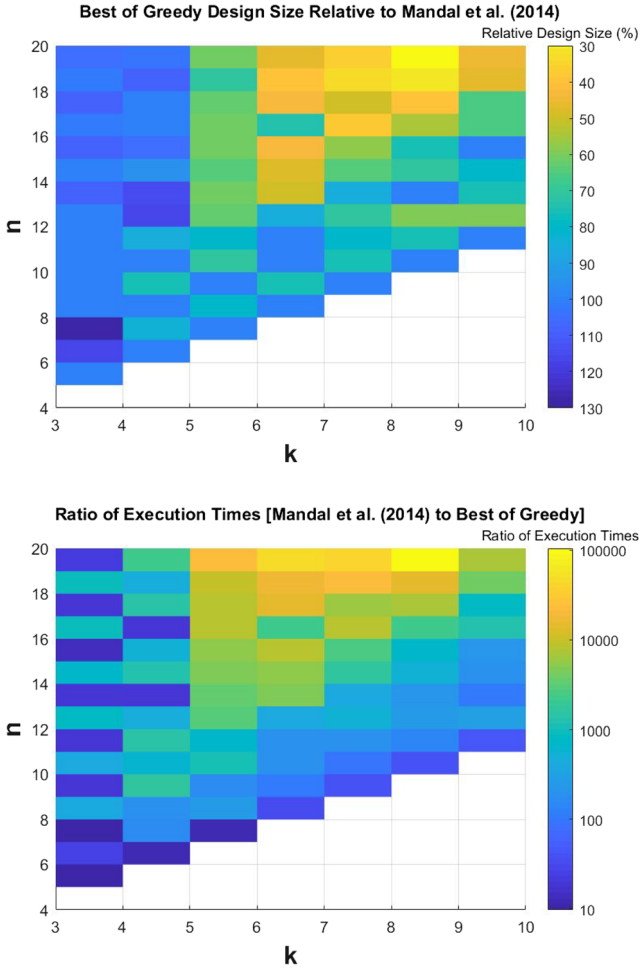

Fig. 6 Best of Greedy versus Mandal et al. (2014) algorithms. The top panel illustrates the size of the Best of Greedy design relative to the corresponding Mandal et al. design. Smaller percentages indicate a greater size discrepancy in favor of Best of Greedy. The lower panel illustrates the ratio of Mandal et al. execution times relative to Best of Greedy. Higher values indicate a greater execution time discrepancy in favor of Best of Greedy.

25% of the size of the Mandal et al. design. In summary, the BoG algorithm produces more efficient experiment designs than did the Mandal et al. algorithm in the large majority of tested cases, and does so in drastically less time.

Discussion

It is worth starting this Discussion by noting what our algorithm does not do: It does not find BIBDs (at least, not deliberately or frequently), and its solutions are not guaranteed to be optimal. Instead, the goal of the algorithm is to produce efficient incomplete block designs for given values of n and k, so that each treatment pair is included in a block at least λ_0 times. The BoG algorithm performs very well in this capacity, providing good (and often superior) designs very quickly for any combination of n and k. We considered candidate algorithms according to their flexibility (ability to produce designs for any given combination of n and k), computational requirements, and efficiency (the number of blocks in the design produced by the algorithm). The Best of Greedy algorithm is an appropriate choice on all counts. This algorithm produced the best solution in more than 99% of the tested combinations of n and k, and it did so very quickly, with a median execution time of less than 1 s using our software implementation and computing platform. We also demonstrated that the BoG algorithm outperformed the widely available Mandal et al. (2014) algorithm in the large majority of tested cases, and it did so orders of magnitude more quickly.

Of course, we do not claim that this approach bests all others for all combinations of n and k, nor do we claim that the BoG algorithm will identify BIBDs when they exist. In fact, when λ_0 is set to 1, there are other examples in the literature of designs with fewer blocks for particular combinations of n and k (e.g., Abel, 1994; Braun, Etzion, Östergård, Vardy, & Wassermann, 2016, Colbourn & Mathon, 1980; Di Paola, Seberry Wallis, & Wallis, 1973; Mathon & Rosa, 1996).[5] For example, when $n = 7$, $k = 3$, and $\lambda_0 = 1$, a BIBD exists with seven blocks, but the BoG algorithm typically produces a design with nine blocks. We encourage investigators to take advantage of these solutions when their experiment design constraints match those of the published designs. However, published solutions exist for only a small subset of the parameter space. If there is no published solution for the researcher's particular given values of n and k, then the BoG algorithm is an attractive alternative that provides efficient, quick solutions to a common experiment design problem. Conveniently, researchers can download our software in order to quickly create efficient incomplete block designs for any combination of treatments and block sizes. Matlab script, Python code, a Windows executable, and a web application to generate experimental designs are available at http://justinmacdonald.net/projects. In fact, we have already taken advantage of our software in order to construct an experimental design for a published research project (Coburn et al., 2019).

Although this was not tested in this article, the algorithm can be generalized quite easily in several ways. For example, it is quite straightforward to extend the algorithm so as to produce designs in which more than one observation per pair is required (i.e., $\lambda_0 > 1$). The algorithm just proceeds assigning treatments to blocks until the required number of concurrences per pair have been included in the design. The Best of Greedy algorithm works well in this situation, as well: Fewer unnecessary duplicate pairs are included in the design. For example, the number of blocks in the design that include two observations per pair will be less than twice the length of the solution that includes only one observation per pair. In addition, it is easy to extend the algorithm in order to consider larger groupings of treatments than the pairs considered here. If one wanted to create a design

with n treatments and k treatments per block and also required that all $\binom{n}{3}$ *triples* needed to appear together at least once, for example, the algorithm would use a three-dimensional concurrence matrix, and proceed as normal otherwise. In addition to generating the designs considered in this article, the Matlab and Python scripts provided on the website also include both of these extensions: the capability to design block lists that require more than one observation of each pair, and the construction of experiment designs focused on the inclusion of treatment groupings other than pairs.

Open Publishing Statement The dataset from the simulation (in Matlab data format) is available for download at https://osf.io/q24yu/?view_only=9bffd261f70a409babcb21991499d763.

References

Abel, R. J. R. (1994). Forty-three balanced incomplete block designs. *Journal of Combinatorial Theory: Series A*, *65*, 252–267. https://doi.org/10.1016/0097-3165(94)90023-X

Berman, M. G., Hout, M. C., Kardan, O., Hunter, M., Yourganov, G., Henderson, J. M., … Jonides, J. (2014). The perception of naturalness correlates with low-level visual features of environmental scenes. *PLoS ONE*, *9*, e114572. https://doi.org/10.1371/journal.pone.0114572

Bose, R. C. (1949). A note on Fisher's inequality for balanced incomplete block designs. *Annals of Mathematical Statistics*, *20*, 619–620.

Braun, M., Etzion, T., Östergård, P. R., Vardy, A., & Wassermann, A. (2016). Existence of q-analogs of Steiner systems. *Forum of Mathematics, Pi*, *4*, E7. https://doi.org/10.1017/fmp.2016.5

Clay, J. R. (1972). Generating balanced incomplete block designs from planar near rings. *Journal of Algebra*, *22*, 319–331.

Coburn, A., Kardan, O., Kotabe, H., Steinberg, J., Hout, M., Robbins, A., … Berman, M. (2019). Psychological responses to natural patterns in architecture. *Journal of Environmental Psychology*, *62*, 133–145. https://doi.org/10.1016/j.jenvp.2019.02.007

Cochran, W. G., & Cox, G. M. (1957). *Experimental designs.* New York, NY: Wiley.

Colbourn, M. J., & Mathon, R. A. (1980). On cyclic Steiner 2-designs. *Annals of Discrete Mathematics*, *7*, 215–253.

Di Paola, J. W., Seberry Wallis, J., & Wallis, W. D. (1973). A list of balanced incomplete block designs for $r < 30$. In *Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory and Computing* (Vol. 8, pp. 249–258). Winnipeg, MB: Congressus Numerantium.

Fisher, R. A., & Yates, F. (1963). *Statistical tables for biological, agricultural and medical research*. Edinburgh, UK: Oliver & Boyd.

Goldstone, R. (1994). An efficient method for obtaining similarity data. *Behavior Research Methods, Instruments, & Computers*, *26*, 381–386. https://doi.org/10.3758/BF03204653

Hanani, H. (1975). Balanced incomplete block designs and related designs. *Discrete Mathematics*, *11*, 255–369.

Hays, W.L. (1994) *Statistics* (5th ed.). Fort Worth, TX: Harcourt Brace.

Horst, J. S., & Hout, M. C. (2015). The Novel Object and Unusual Name (NOUN) Database: A collection of novel images for use in experimental research. *Behavior Research Methods*, *48*, 1393–1409. https://doi.org/10.3758/s13428-015-0647-3

Hout, M. C., Cunningham, C., Robbins, A., & MacDonald, J. (2018). Simulating the fidelity of data for large stimulus set sizes and variable dimension estimation in multidimensional scaling. *Sage Open*, *8*(2). https://doi.org/10.1177/2158244018773143

Hout, M. C., Godwin, H. J., Fitzsimmons, G., Robbins, A., Menneer, T., & Goldinger, S. D. (2015). Using multidimensional scaling to quantify similarity in visual search and beyond. *Attention, Perception, & Psychophysics*, *78*, 3–20. https://doi.org/10.3758/s13414-015-1010-6

Hout, M. C., Goldinger, S. D., & Brady, K. J. (2014). MM-MDS: A multidimensional scaling database with similarity ratings for 240 object categories from the Massive Memory picture database. *PLoS ONE*, *9*, e112644. https://doi.org/10.1371/journal.pone.0112644

Hout, M. C., Goldinger, S. D., & Ferguson, R. W. (2013). The versatility of SpAM: A fast, efficient spatial method of data collection for multidimensional scaling. *Journal of Experimental Psychology: General*, *142*, 256–281. https://doi.org/10.1037/a0028860

Hout, M. C., & Goldinger, S. D. (2016). SpAM is convenient, but also satisfying: Reply to Verheyen et al. (2016). *Journal of Experimental Psychology: General*, *3*, 383–387. https://doi.org/10.1037/xge000014

Hout, M. C., Papesh, M. H., & Goldinger, S. D. (2012). Multidimensional scaling. *Wiley Interdisciplinary Reviews: Cognitive Science*, *4*, 93–103. https://doi.org/10.1002/wcs.1203

Kimmel, H. D., Becker, J. A., & Bevill, M. J. (1988). Balanced incomplete blocks to control individual differences. *Behavior Research Methods, Instruments, & Computers*, *20*, 301–312.

Mandal, B. N. (2019). ibd: Incomplete block designs. R package version 1.5. Retrieved from https://CRAN.R-project.org/package=ibd

Mandal, B. N., Gupta, V. K., & Parsad, R. (2014). Efficient incomplete block designs through linear integer programming. *American Journal of Mathematical and Management Sciences*, *33*, 110–124.

Mathon, R., & Rosa, A. (1996). 2-(v, k, λ) designs of small order. In C. J. Colbourn (Ed.), *The CRC handbook of combinatorial designs* (pp. 25–57). New York, NY: Taylor & Francis.

R Core Team. (2013). R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org/

Vazirani, V. V. (2003). *Approximation algorithms.* Berlin, Germany: Springer.