Using anticlustering to partition data sets into equivalent parts

Martin Papenberg & Gunnar W. Klau

Heinrich Heine University Düsseldorf

Author note

Martin Papenberg, Department of Experimental Psychology, Heinrich Heine University Düsseldorf; Gunnar W. Klau, Department of Computer Science, Heinrich Heine University Düsseldorf.

Part of the work presented in this paper is based on Martin Papenberg's Bachelor thesis at the Computer Science Department at Heinrich Heine University Düsseldorf, supervised by Gunnar W. Klau. The software package `anticlust`, which is presented in this paper, was presented at the TeaP conference in London in April, 2019, and at a meeting at the Department of Experimental Psychology at Heinrich Heine University Düsseldorf in July, 2019. The slides presented at these occasions can be retrieved from https://osf.io/cd5sr/. The documentation accompanying the software package `anticlust` also contains some background on the anticlustering methodology, retrieved from https://github.com/m-Py/anticlust. The preprint of this paper can be retrieved from https://psyarxiv.com/3razc/. The open science framework contains supplementary data and analysis scripts, available via https://doi.org/10.17605/OSF.IO/CD5SR.

We would like to thank Juliane V. Tkotz for her detailed feedback on an earlier version of this manuscript.

Correspondence concerning this article should be addressed to Martin Papenberg, Heinrich Heine Universität Düsseldorf, Institut für Experimentelle Psychologie, Universitätsstraße 1, 40225 Düsseldorf, Germany. E-mail: martin.papenberg@hhu.de

Abstract

Numerous applications in psychological research require that a pool of elements is partitioned into multiple parts. While many applications seek groups that are well-separated, i.e., dissimilar from each other, others require the different groups to be as similar as possible. Examples include the assignment of students to parallel courses, assembling stimulus sets in experimental psychology, splitting achievement tests into parts of equal difficulty, and dividing a data set for cross validation. We present `anticlust`, an easy-to-use and free software package for solving these problems fast and in an automated manner. The package `anticlust` is an open source extension to the `R` programming language and implements the methodology of anticlustering. Anticlustering divides elements into similar parts, ensuring similarity between groups by enforcing heterogeneity within groups. Thus, anticlustering is the direct reversal of cluster analysis that aims to maximize homogeneity within groups and dissimilarity between groups. Our package `anticlust` implements two anticlustering criteria, reversing the clustering methods k-means and cluster editing, respectively. In a simulation study, we show that anticlustering returns excellent results and outperforms alternative approaches like random assignment and matching. In three example applications, we illustrate how to apply `anticlust` on real data sets. We demonstrate how to assign experimental stimuli to equivalent sets based on norming data, how to divide a large data set for cross validation, and how to split a test into parts of equal item difficulty and discrimination.

Keywords: anticlustering, maximum diverse grouping problem, cluster editing, k-means clustering, partitioning

Word count: 8000

Using anticlustering to partition data sets into equivalent parts

Numerous applications in psychological research require that a pool of elements is partitioned into multiple parts, while ensuring that all parts are as similar as possible (Brusco, Cradit, & Steinley, in press). When dividing students into groups in a schooling context, teachers are often interested in assembling groups that are similar with regard to scholastic ability and demographic composition (Desrosiers, Mladenović, & Villeneuve, 2005). In educational psychology, it is sometimes necessary to split a pool of test items into parts of equal length that are presented to different cohorts of students. To ensure test fairness, it is required that all parts are equally difficult (Brusco, Köhn, & Steinley, 2013; Gierl, Daniels, & Zhang, 2017; van der Linden, 2005). Splitting a data set also has applications in the context of data analysis: With the increasing demand for machine learning methods in psychology, cross validation techniques have become more important to test the predictive performance of statistical models (Yarkoni & Westfall, 2017). Most cross validation require that a data set is split into multiple parts where one part serves as a validation sample. While the data is usually partitioned using a random split, it is preferable to employ a partitioning method that balances the distribution of the variables in the data set across samples (e.g., Zeng & Martinez, 2000).

Another important partitioning problem arises in experimental psychology regarding the assembly of stimulus sets. When designing a study, researchers are often faced with the challenge to select multiple subsets of stimuli that are presented in different experimental conditions (Brusco & Stahl, 2001). When the experimental manipulation is realized within-subjects, it is desirable that the stimulus sets accompanying the different experimental conditions are as similar as possible on dimensions that affect the participants' responses; differences between conditions should be attributable to the experimental manipulation and not to the materials (Lahl & Pietrowsky, 2006). For example, Lahl, Wispel, Willigens, and Pietrowsky (2008) investigated the effect of napping on recall memory.

In their study, each participant completed a napping session and a wake session, separated by a one week wash out period. Before each session, participants had to memorize a list of 30 words; after each session, word recall was tested. Due to possible carry-over effects, presenting the same word list in both conditions was not feasible. Instead, two word lists had to be created and counterbalanced across the experimental conditions (wake versus sleep). It was desirable that both lists were as similar as possible on dimensions that are known to affect memory performance.

Researchers have usually relied on ad-hoc approaches when splitting a stimulus pool into similar parts. Mostly, researchers have relied on manual assignment (van Casteren & Davis, 2007), random assignment, and, to a lesser degree, matching-based assignment (e.g., Buchner & Wippich, 2000). Schaper et al. (2019a, 2019b) selected 96 words from a total pool of 1,004 words based on a desired range in typicality ratings. Next, they manually partitioned the item pool into three equal-sized sets in such a way that mean values of typicality, number of syllables, and word frequency were similar. Unfortunately, manually selecting stimulus sets is time consuming and daunting, and the resulting sets are usually not optimally matched with regard to their similarity (Armstrong, Watson, & Plaut, 2012; Cutler, 1981; van Casteren & Davis, 2007). Other researchers have randomly assigned stimuli to experimental sets (e.g., Heck & Erdfelder, 2016; Kroneisen & Bell, 2018). While random assignment may possess face validity as a partitioning procedure, it does not guarantee that the resulting sets will be similar. Lahl and Pietrowsky (2006) proposed to match stimuli based on a mathematical distance between quantified stimulus properties—such as word frequency or typicality ratings—and advocated the following procedure to assign stimuli to experimental sets: (a) compute the pairwise distances between all items; (b) match the two most similar items on the basis of the distance measure; (c) assign the matched items to different sets; repeat steps (b) and (c) until the experimental sets are filled. They did not specify a rule for step (c), i.e., how to assign the matched items to sets, leaving this decision to the researcher. Usually, the matched stimuli would be

allocated to sets via random assignment (e.g., Buchner & Wippich, 2000).

Matching is not well-suited for partitioning a stimulus pool into similar parts. Figure 1 illustrates a problem that matching-based partitioning may run into. Here, four stimuli described by two numeric attributes, x and y, have to be allocated to two sets. The matching will pair the two items in the upper left and the two items in the lower right of the plot. Two partitionings are possible based on the matching, shown in the middle and right panel of Figure 1. Of these two, partitioning (b) is clearly to be preferred because in partitioning (a), one stimulus set consists of stimuli with larger values on both numeric attributes. Unfortunately, matching does not prevent—and may instead produce—such misassignments. Matching does not differentiate between the different possible ways to allocate stimuli after the matching has been conducted. In the present case of only four stimuli, a researcher might be able to manually correct the assignment. However, with many stimuli and many numeric attributes, a manual correction quickly becomes impossible (Cutler, 1981). The number of possible assignments after the matching grows exponentially with the size of the stimulus pool. Therefore, intrinsically, partitioning an item pool into similar parts is not a matching problem. A different approach is needed that unambiguously strives to maximize the similarity of the stimulus sets.

Partitioning a stimulus pool into parts resembles a clustering application because a pool of elements has to be divided into multiple disjunct subsets, which is accomplished by cluster analysis (Rokach & Maimon, 2005). However, a cluster analysis creates subsets in such a way that the elements within clusters are homogeneous—i.e., similar to each other—but dissimilar from the elements in other clusters. When partitioning a stimulus pool into experimental sets, the opposite is desired: the different sets should be as similar as possible. Interestingly, Späth (1986) and Valev (1983, 1998) already noted that by reversing the logic of the popular k-means clustering method, they were able to establish clusters that were similar to each other. They independently coined the term *anticlustering* for this

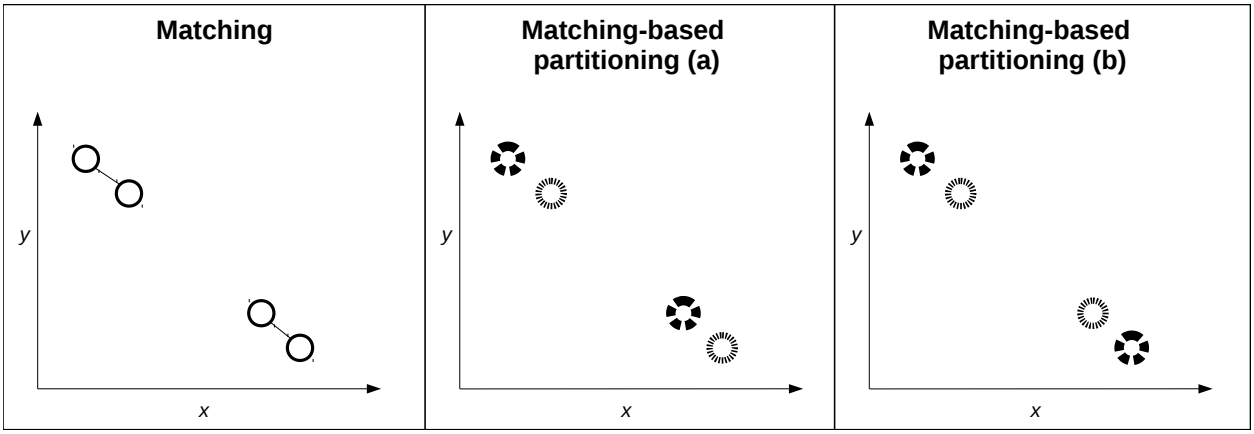

Figure 1. An illustration of matching-based partitioning. Based on the identification of similar item pairs (matches), a stimulus pool of four items has to be partitioned into two sets that should be as similar as possible on two numeric attributes.

purpose. Notably, it has been recognized that forming similar groups is equivalent to maximizing the heterogeneity within groups (Feo & Khellaf, 1990). That is, anticlustering ensures similarity *between* groups by enforcing dissimilarity—i.e., heterogeneity—*within* groups. Thus, anticlustering diametrically reverses the logic of cluster analysis that strives to maximize homogeneity within groups and dissimilarity between groups. Correspondingly, anticlustering has also been referred to as the "maximally diverse grouping problem" (e.g., Fan, Chen, Ma, & Zeng, 2011; Brusco et al., in press). A problem that has close resemblance to the maximum diverse grouping problem is the "maximum diversity problem" (Glover, Kuo, & Dhir, 1998). However, the maximum diversity problem requires to extract only a subset of all elements in such a way that these are as diverse as possible, whereas anticlustering allocates each element to a group. A variety of other objectives have also been discussed as suitable anticlustering criteria (Baker & Powell, 2002).

The need for anticlustering methods has been recognized, among others, in the areas of operational research (e.g., Baker & Powell, 2002; Palubeckis, Ostreika, & Rubliauskas, 2015), management science (e.g., Krass & Ovchinnikov, 2006), and artificial intelligence (e.g., Steghöfer, Behrmann, Anders, Siefert, & Reif, 2013). The problem has also been studied

with emphasis on particular applications, such as very large-scale integration (Weitz & Lakshminarayanan, 1997, 1998), working or student group formation (Baker & Powell, 2002; Krass & Ovchinnikov, 2006), and exam scheduling (Weitz & Lakshminarayanan, 1997). Despite the wide applicability, the value of anticlustering has not been recognized in psychology up until recently (cf. Steinley, 2006). This underutilization most likely stems from two driving factors: First, a formal introduction of anticlustering to the psychological literature was only provided very recently by Brusco et al. (in press). Thus, it is likely that most psychologists were not aware that some of their research problems could be solved automatically via anticlustering. Second, and maybe more severely, anticlustering methods have previously not been accessible through free or commercial software packages. Thus, even if some psychologists were aware of the benefits of anticlustering, they would have required a certain programming skill set—and a substantial amount of time—to implement these methods themselves.

Our primary contribution is to make the anticlustering methodology more accessible to researchers in psychology. First, we provide a formal introduction to the theory of anticlustering, focusing on two major anticlustering objectives: k-means clustering (Jain, 2010; Steinley, 2006) and cluster editing (Shamir, Sharan, & Tsur, 2004; Zahn, 1964). Second and most critically, we present `anticlust`, a free and easy-to-use software implementation of anticlustering. It is available as an open source software extension to the widely used statistical programming language `R` (R Core Team, 2019). Using real and simulated data, we demonstrate the superiority of anticlustering in comparison to other partitioning strategies that researchers have employed (manual assignment, random assignment, matching). We conclude by providing several examples of how to apply `anticlust` on real data sets. The examples include the creation of stimulus sets in experimental psychology, splitting a large data set for cross validation, and splitting a test into parts of equal difficulty.

Problem Formalization

In anticlustering, an item pool $X = \{x_1, \ldots, x_N\}$ has to be partitioned into K subsets $C = \{c_1, \ldots, c_K\}$. We will refer to the subsets c_j, $j = 1, \ldots, K$, as anticlusters, sets, or groups, and to the collection of anticlusters C as the anticlustering partitioning. We assume that each item x_i, $i = 1, \ldots, N$, is a vector of length M where each entry describes one of its numeric attributes. Thus, the data input X is interpreted as an $N \times M$ matrix D where each row represents an item x_i (e.g., representing a stimulus or a person) and each column is a numeric attribute (e.g., the number of syllables in a word, or the attractiveness rating of a face). For the sake of simplicity, we will only use a positional index $i = 1, \ldots, N$ to refer to an item x_i henceforth.

The anticlustering partitioning has to satisfy the following restrictions:

$$\bigcup_{j=1}^{K} c_j = X \tag{1}$$

$$c_j \cap c_k = \emptyset,\ \forall j, k \in \{1, \ldots, K\},\ j \neq k \tag{2}$$

$$|c_j| = |c_k|,\ \forall j, k \in \{1, \ldots, K\} \tag{3}$$

Restriction (1) ensures that each element from the pool is assigned to an anticluster; restriction (2) ensures that each element is assigned to only one anticluster; restriction (3) ensures that each anticluster contains the same number of elements, namely $\frac{N}{K}$. Enforcing the same number of elements in each set follows from the convention that in most anticlustering applications—such as constructing conditions in experimental psychology—groups should be as parallel as possible; it is not a general restriction for anticlustering methods.

The anticlustering partitioning C has to be chosen in such a way that all groups are as similar as possible. Similarity is assessed on the basis of the data input D for a given partitioning. The following sections detail how between-group similarity can be computed in

the context of anticlustering. In particular, we discuss two important anticlustering criteria, corresponding to the clustering methods k-means and cluster editing.

K-means clustering

The popular k-means clustering method aims at minimizing the within-group variance, i.e., the sum of the squared Euclidean distances between each data point and its cluster center (Jain, 2010):

$$Var_{\text{within}} = \sum_{j=1}^{K} \sum_{i \in c_j} ||i - \mu_j||^2 \tag{4}$$

The cluster center μ_j is composed of the mean feature values in a given cluster $c_j, j \in 1, \ldots, K$ (illustrated by the triangulars in Figure 2A and 2B); $||i - j||$ is the Euclidean distance between two items i and j, defined as the root of the sum of the squared differences between items on each feature:

$$d_{\text{eucl}}(i, j) = ||i - j|| = \sqrt{\sum_{m=1}^{M} (i_m - j_m)^2} \tag{5}$$

When items are described by two features (i.e., $M = 2$), the Euclidean distance corresponds to the geometric, "straightline" distance between points in a two-dimensional space; more similar items are closer to each other (see Figure 2).

Späth (1986) and Valev (1983, 1998) proposed to maximize the variance criterion to create similar sets, coining the term anticlustering. Späth (1986) showed that maximimizing the variance criterion directly minimizes differences between the cluster centers. When the within-cluster variance is maximal, the cluster centers become as similar as possible (see Figure 2B).

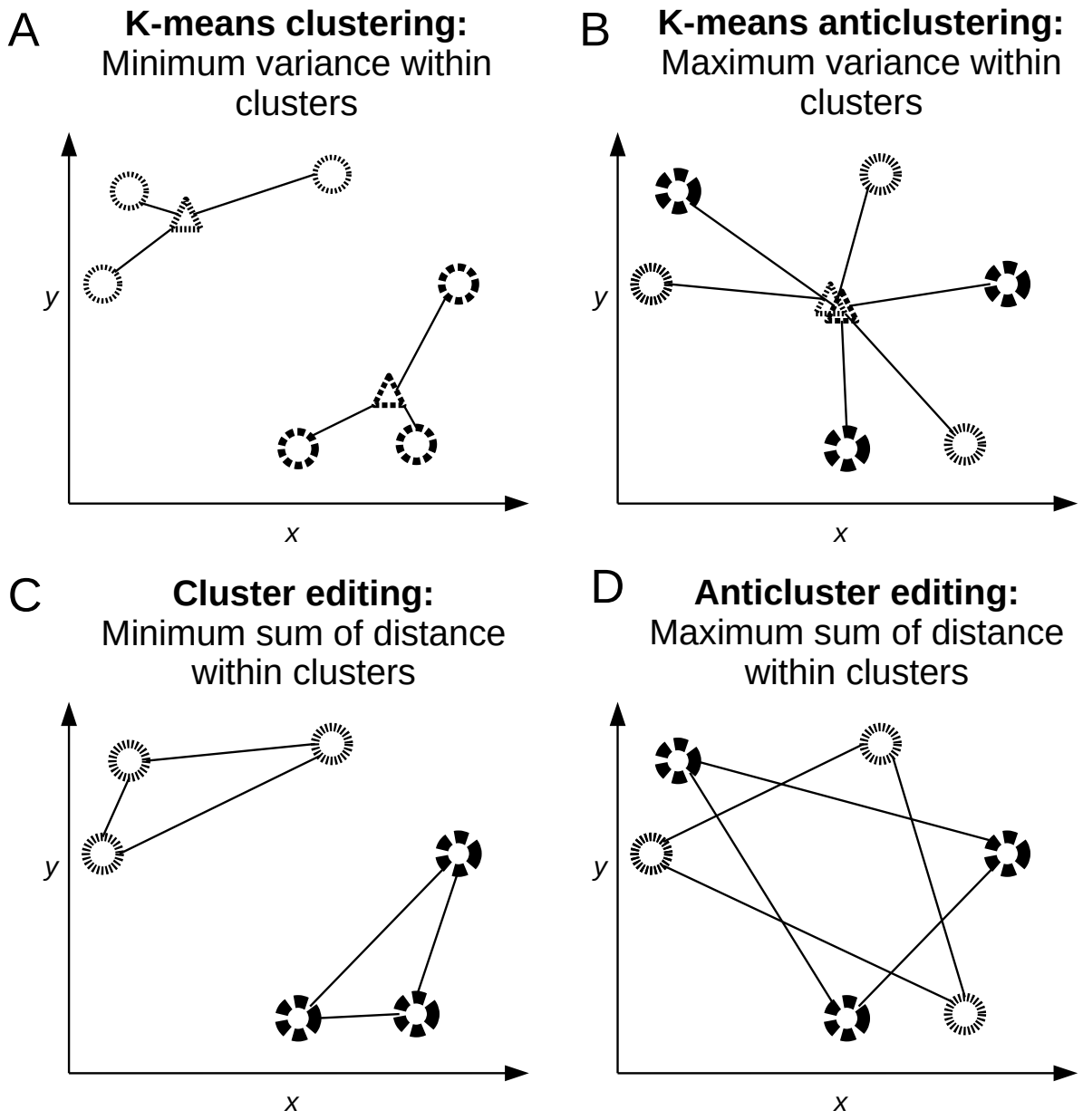

Figure 2. For the purpose of illustration, six items, described by two numeric features x and y, have been partitioned into two equal-sized groups. A cluster partitioning is obtained when the within-cluster heterogeneity is low and the between-cluster heterogeneity is high (panels A and C). An anticluster partitioning is obtained when the within-cluster heterogeneity is high and the between-cluster heterogeneity is low (panels B and D). K-means clustering measures within-cluster heterogeneity as the sum of squared Euclidean distances between data points and cluster centers (shown as the triangulas in panels A and B). Cluster editing measures within-cluster heterogeneity as the sum of pairwise dissimilarities within groups (illustrated in panels C and D). In each panel, the solid lines illustrate the distances that enter the objective functions of the respective clustering methods.

Cluster editing

Another popular anticlustering method reverses a different clustering paradigm, namely cluster editing (Shamir et al., 2004; Zahn, 1964). Cluster editing has also been studied under different names such as correlation clustering (Bansal, Blum, & Chawla, 2004), clique partitioning (Grötschel & Wakabayashi, 1989), and transitivity clustering (Wittkop et al., 2010). Brusco and Köhn (2009) were the first to treat cluster editing as an analytic tool for researchers in psychology. K-balance partitioning, which is closely related to cluster editing, has been studied by Brusco and Steinley (2010).

Cluster editing is based on a notion of pairwise dissimilarity between the units of analysis. In particular, cluster editing requires to categorize pairs of elements into similar versus dissimilar. Mathematically, for an index d_{ij}, $d_{ij} < 0$ implies that two elements i and j are similar, whereas $d_{ij} > 0$ implies they are dissimilar. The higher the absolute value of d_{ij}, the more (dis)similar are the elements i and j. In the *unweighted* variant of cluster editing, there is only a binary distinction, $d_{ij} = -1$ implying that i and j are similar and $d_{ij} = 1$ implying they are dissimilar. If d_{ij} refers to an index of similarity rather than dissimilarity, the interpretation is reversed.[1]

The cluster editing objective function, which should be minimized, is given as the as the sum of pairwise dissimilarities between elements within the same cluster (see Figure 2C and 2D):

$$D_{\text{within}} = \sum_{1 \leq i < j \leq n} d_{ij} x_{ij} \tag{6}$$

[1] While the cluster editing literature usually uses measures of similarity rather than dissimilarity, we discuss dissimilarities because this case is more common in the anticlustering literature. Note that both cases are equivalent: minimizing dissimilarity within clusters leads to the same results as maximizing similarity within clusters, if an index of dissimilarity is converted to an index of similarity by multiplying it by -1.

The variables x_{ij} encode whether two items i and j are part of the same anticluster:

$$x_{ij} = \begin{cases} 1 & \text{if } x_i \in c_k \wedge x_j \in c_k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

When minimizing D_{within}, the data is partitioned into homogenous clusters: elements within each cluster tend to be similar and elements in different clusters tend to be dissimilar (Figure 2C). For the reversed anticluster editing problem, the within-cluster sum of dissimilarities D_{within} has to be maximized (Figure 2D). An appealing feature of cluster editing is that the optimal number of clusters is identified solely based on the pairwise dissimilarities, whereas most other clustering methods require that the number of clusters is specified by the researcher (Brusco & Köhn, 2009). However, this features crucially hinges on the quality of the dissimilarity measure. In particular, it depends on the correctness of the dissimilarity classification, i.e., setting $d_{ij} < 0$ if i and j are truly similar and $d_{ij} > 0$ if i and j are truly dissimilar. While such a classification may be difficult with regard to the relationship between psychological variables, the detection of the correct number of clusters crucially depends on it.

Fortunately, the restrictions on the dissimilarities d_{ij} are relaxed in the anticlustering application. There is no need to classify pairs of items as similar versus dissimilar, which, in the case of cluster editing, is necessary to automatically deduce the number of clusters. In anticlustering, the number of anticlusters and the size thereof is usually fixed by the application and in most cases, anticlusters of equal size are required (Gallego, Laguna, Marti, & Duarte, 2013). Since the number of groups is specified a priori, it is sufficient to employ a monotonous measure of dissimilarity where larger values indicate higher dissimilarity. In principle, d_{ij} may refer to any theoretically sound measure of object dissimilarity (e.g., Dry & Storms, 2009; Nosofsky, 1992). For many practical purposes, d_{ij} will refer to the Euclidean

distance (e.g., Gallego et al., 2013), which is also the default option in our software `anticlust`. However, note that a variety of dissimilarity measures have been proposed to quantify stimulus dissimilarity that may be preferable depending on the researcher's needs (e.g., Tversky, 1977).

The sum D_{within} quantifies the diversity within groups as the sum of dissimilarities within groups; a larger value of D_{within} implies that the within-group heterogeneity is high. At the same time, maximizing D_{within} minimizes the sum of dissimilarities between elements that are in different groups (Baker & Powell, 2002; Feo & Khellaf, 1990). Therefore, maximizing D_{within} minimizes the average dissimilarity[2] between items in different groups, making D_{within} a suitable measure of between-group similarity. Because D_{within} is also a measure of the overall diversity in a given partitioning, anticluster editing has usually been referred to as the maximally diverse grouping problem (e.g., Fan et al., 2011; Brusco et al., in press; Gallego et al., 2013; Palubeckis et al., 2015; Urošević, 2016). Closely tied to the problem of maximum diversity is the problem of maximum dispersion, which refers to the minimum value of dissimilarity between two elements within a cluster. Thus, dispersion is a measure of "worst-case" pairwise dissimilarity across all clusters (Brusco et al., in press). In applications that require high within-group diversity, ensuring a high dispersion is crucial. Brusco et al. (in press) recently provided a novel algorithm that aims to simultaneously maximize D_{within} and the dispersion. Since we are more interested in maximizing between-group similarity rather than within-group heterogeneity, it is sufficient for us to focus on maximizing just D_{within}.

Solution Methods

To solve anticlustering in an automated manner, we implemented heuristic and exact algorithms that maximize the criteria Var_{within} (k-means anticlustering) and D_{within}

[2] Due to the restriction that each stimulus set has the same size, minimizing the sum of distances is equivalent to minimizing the average distance between sets.

(anticluster editing). Unfortunately, finding a partitioning that maximizes similarity according to these criteria is computationally challenging. In particular, cluster editing, anticluster editing, as well as k-means clustering have been shown to be NP-hard (Aloise, Deshpande, Hansen, & Popat, 2009; Feo & Khellaf, 1990; Shamir et al., 2004), implying that it is highly unlikely to find an exact algorithm that always returns the optimal anticlustering partitioning in acceptable running time, at least when N is large. When N increases, the number of anticlustering partitionings increases exponentially (see Figure 3), quickly rendering it impossible for a computer to try them all out in acceptable running time. This is true regardless of the speed with which a single solution can be investigated (Garey & Johnson, 1979; van Rooij, 2008). NP-hard problems are usually tackled using heuristic algorithms that—while not guaranteeing optimality—generally return satisfying results (Karp, 1986). While exact algorithms are not often used in practice, they serve as useful benchmarks for the performance of heuristics. Therefore, we implemented an exchange-based heuristic as well as exact algorithms based on integer linear programming.

Exchange Method

Our heuristic algorithm adapts an exchange procedure proposed by Weitz and Lakshminarayanan (Method LCW; 1998; also see Gallego et al., 2013; Späth, 1986). The exchange method is based on swapping items between anticlusters in such a way that each swap improves the objective value by the largest possible margin. It proceeds as follows: First, arbitrarily assign items to anticlusters, while ensuring that each anticluster consists of the same number of items. Then, select the first item and check how the objective function would change if the item was swapped with each item that is currently assigned to a different anticluster. After simulating each possible exchange—a total of $(N - \frac{N}{k})$ exchanges—realize the one exchange that increases the objective the most. No exchange is realized if the objective cannot be improved. The algorithm terminates when the exchange process has been repeated for each item.

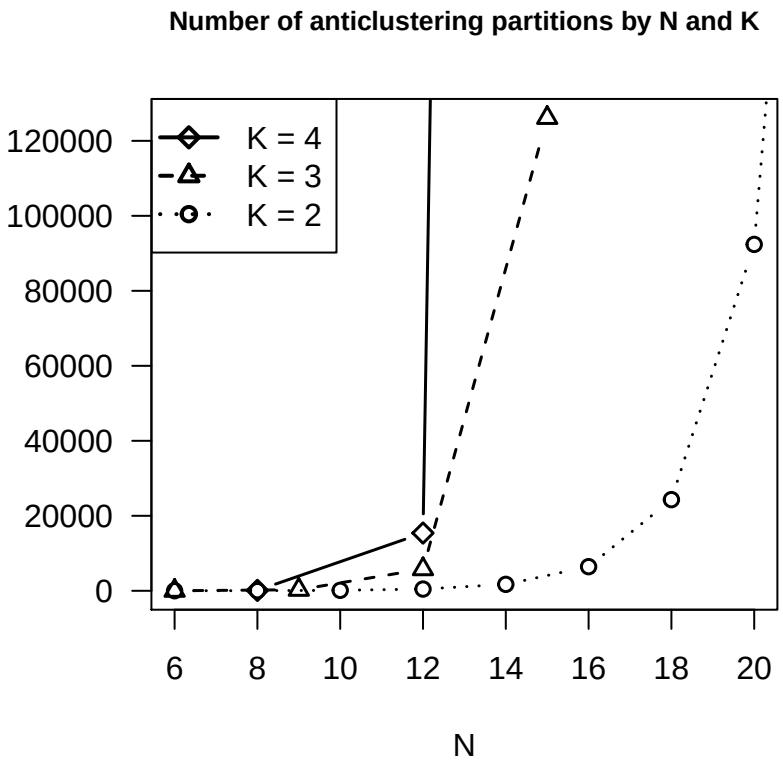

Figure 3. Illustrates the exponential increase of anticlustering partitionings with increasing N and varying K. The number of ways to partition a set of N elements into K equal-sized subsets is given as $\frac{1}{K!} \cdot \prod_{i=0}^{K-1} \binom{N-\frac{iN}{K}}{\frac{N}{K}}$.

Note that the exchange algorithm presented here is very general; it can be used to optimize any objective function quantifying group similarity, not just the anticlustering objectives Var_{within} or D_{within}. We therefore allow the user in our software `anticlust` to define customized objective functions that are optimized using the exchange method. This general architecture makes it easy to adapt additional anticlustering objectives in the future (e.g., Brusco et al., in press; Baker & Powell, 2002).

Exact Anticluster Editing

We implemented exact solution methods on the basis of integer linear programming. Integer linear programming is a very general algorithmic framework used to solve computationally intractable problems such as anticluster editing to provable optimality (Nemhauser & Wolsey, 1988). Integer linear programming often dramatically improves the time needed to find an optimal solution as compared to a complete enumeration of the entire solution space (e.g., Böcker, Briesemeister, & Klau, 2011; Grötschel & Wakabayashi, 1989). Enumerating all possible anticlustering partitionings would also guarantee to find an optimal solution, but this approach already becomes infeasible even for very small problem sizes (cf. Figure 3). In contrast, integer linear programming is a more "clever" technique that usually precludes the necessity to enumerate the entire solution space. The degree to which running time is improved depends on the nature of the specific problem (Chen, 2017).

While integer linear programming is heavily relied on in the field of operational research, psychological applications have been scarce. Nevertheless, there have been important adaptions of integer linear program models. For example, van der Linden (2005) proposed numerous models for psychological test design. Brusco and Stahl (2001) presented some integer linear programming models for extracting subsets of stimuli from confusion matrices. These authors also used integer linear programming to select multiple subsets of stimuli, such there was high similarity within groups and low similarity between groups, which is the opposite of the anticlustering problem (also see Brusco & Steinley, 2006; Brusco et al., 2013).

Our integer linear program extends a formulation that Grötschel and Wakabayashi (1989) provided to optimally solve cluster editing. The model identifies values for binary variables x_{ij}—representing whether each pair of items is part of the same anticluster—that maximize the objective function D_{within}. The optimization is subject to constraints on the values the binary variables are allowed to assume, implemented as linear inequalities:

$$-x_{ij} + x_{ik} + x_{jk} \leq 1, \qquad \forall\, 1 \leq i < j < k \leq N, \tag{8}$$

$$x_{ij} - x_{ik} + x_{jk} \leq 1, \qquad \forall\, 1 \leq i < j < k \leq N, \tag{9}$$

$$x_{ij} + x_{ik} - x_{jk} \leq 1, \qquad \forall\, 1 \leq i < j < k \leq N, \tag{10}$$

$$\sum_{1 \leq i < j \leq N} x_{ij} + \sum_{1 \leq k < i \leq N} x_{ki} = \frac{N}{K} - 1, \qquad \forall i \in \{1, ..., N\} \tag{11}$$

$$x_{ij} \in \{0, 1\},\ \forall\, 1 \leq i < j \leq N \tag{12}$$

Constraints (8) to (10) ensure that only items within the same anticluster are connected, i.e., $x_{ij} = 1$ only if items i and j are part of the same anticluster, and 0 otherwise. Constraint (11) ensures that K groups of cardinality $\frac{N}{K}$ are returned, by enforcing that each item is clustered with exactly $\frac{N}{K} - 1$ other items. This constraint is an addition to the formulation by Grötschel and Wakabayashi (1989) who did not specify the cluster size a priori. Constraint (12) ensures that the decision variables x_{ij} are binary. The objective function and the constraints are represented as a system of linear inequalities that is passed to an integer linear programming solver. An integer linear programming solver acts as a "black box" that is guaranteed to return an optimal solution. Based on the binary variables x_{ij}, the anticluster affiliation of each item is returned as the output of our program. That is, the output is a vector of length N where each entry is an integer between 1 and K.

Preclustering

Even though integer linear programming improves the running time by a vast amount as compared to a brute force search, it cannot escape the potentially exponential running time that eventually dooms all NP-hard problems. To increase the problems sizes that the integer linear programming technique can be applied to, `anticlust` can employ a preprocessing step that we call preclustering. Before the anticlustering procedure, a cluster editing analysis is performed identifying small groups of items that are as similar as possible

to each other. In particular, if two experimental sets have to be created (i.e., $K = 2$), the preclustering step identifies pairs of similar items, i.e., matches. For $K = 3$, preclustering identifies triplets of similar items etc. The preclustering step is formalized using the same integer linear program as the anticluster editing formulation in (8) to (12), with two exceptions. First, the criterion D_{within} has to be minimized and not maximized to obtain homogenous groups or, equivalently, inter-item dissimilarities are multiplied by -1 before maximization. Second, constraint (11) is replaced to ensure that each item is clustered together with $K - 1$ other items:

$$\sum_{1 \leq i < j \leq N} x_{ij} + \sum_{1 \leq k < i \leq N} x_{ki} = K - 1, \qquad \forall i \in \{1, ..., N\} \tag{13}$$

After the cluster editing integer linear program has been processed to identify preclusters, the anticluster editing integer linear program is processed, whereby preclustered items are prevented from joining the same anticluster. Heuristically, assigning these very similar items to a different anticluster ensures that the between-cluster distance becomes small, i.e., preclustering by itself is a useful heuristic to establish similar sets. Note that an optimal solution might nevertheless be prevented by preclustering, as the optimal partitioning might—maybe counterintuitively—require to group preclustered items within the same anticluster. However, because only very similar items are forbidden from joining the same anticluster, preclustering usually does not impair the quality of a solution.

The preclustering constraints reduce the anticlustering problem space considerably because each anticlustering partitioning that would join two preclustered items is no longer feasible. Therefore, running time is improved and larger problem instances can be processed. In the integer linear programming framework, the preclustering restrictions can be enforced by adjusting the dissimilarity matrix in such a way that the distance between any preclustered items is set to $-\infty$. Because anticluster editing is a maximization problem, the

optimal solution can no longer be obtained when preclustered items are part of the same anticluster; any solution where preclustered items are kept in separate sets will have a better objective value according to the criterion D_{within}. The integer linear programming solver is able to exclude all solutions that violate preclustering constraints and browses the remaining solution space more quickly.

Evaluation

We evaluated our anticlustering implementation with regard to running time and the quality of the partitionings it returns. Our analyses used R (Version 3.4.4; R Core Team, 2018) and the R packages `anticlust` (Version 0.3.0; Papenberg, 2019), `dplyr` (Version 0.8.3; Wickham, François, Henry, & Müller, 2019), `papaja` (Version 0.1.0.9842; Aust & Barth, 2018), and `Rcplex` (Version 0.3.3; Bravo & Theussl, 2016). To reproduce all analyses, all code and data can be retrieved from the open science framework (OSF) repository accompanying this manuscript (Papenberg & Klau, 2019).

Running Time

To test if the exact integer linear programming approach is practically feasible for finding an optimal anticlustering partitioning, we determined the empirical running time of anticluster editing. Data sets with varying size between 10 and 100 elements were sampled from a standard normal distribution. Each data set was partitioned into 2 parts (i.e., K was set to 2). The number of features per item was also set to 2. We compared our three algorithms: (a) exact integer linear programming, (b) integer linear programming employing preclustering restrictions and (c) the exchange method. All algorithms were used as implemented in our R package `anticlust`. ILOG CPLEX (Version 12.8.0; IBM, 2019) was used as the backend integer linear programming solver. All tests were running on an Intel Core i7-7700 computer (3.60GHz x 8) with 8 GB RAM running Ubuntu 16.04 LTS. The time limit per test was set to four hours.

Table 1 illustrates the results. The exact integer linear programming approach was only used for $N \leq 30$ because the exponential explosion of the running time already set in; with $N = 30$, two sets were created in about three hours. When including preclustering restrictions, the integer linear programming technique could be applied to data sets up to $N = 70$ in similar time. The exchange method was fast for all of the tested problem sizes as each data set was processed in less than a second.

Simulation Study

In a simulation study, we evaluated anticlustering and other approaches with regard to their ability to create similar sets. As competitors of anticlustering, we chose random assignment and a matching based algorithm following Lahl and Pietrowsky (2006). The random assignment method was realized by simply assigning each stimulus to a set at random. The matching algorithm was specified as follows: We determined the pairwise Euclidean distance between all stimuli, selected the two most similar items, and randomly allocated each of the matched items to a different set. The matched items were removed from the pool and the procedure was repeated until each item was part of a set.

In total, 10,000 simulation runs were conducted: 5,000 runs with $K = 2$ (i.e., the data was split into two equal-sized parts) and 5,000 runs with $K = 3$. Data was generated as follows: A stimulus was defined by a collection of numeric features. The number of stimuli and the number of features varied between simulation runs. The number of stimuli (N) was randomly varied between 10 and 100 while N was always a multiple of K to ensure that the stimulus pool could be evenly split into K parts of size $\frac{N}{K}$. The number of features was randomly varied between 1 and 4. For each run, the distribution of each feature was randomly determined to be either (a) uniform in [0, 1], (b) a standard normal distribution with $M = 0$ and $SD = 1$, or (c) a wider normal distribution with $M = 0$ and $SD = 2$. In a given simulation run, all features were drawn from the same distribution. Features were generated independently, i.e., no correlation between features was modeled. In each run of

Table 1

Running time of the three anticlustering algorithms in dependence of N (K = 2).

N	Exact ILP	ILP/Preclustering	Exchange Method
10	0.06	0.02	< 0.01
12	0.10	0.03	< 0.01
14	0.12	0.05	< 0.01
16	0.28	0.08	< 0.01
18	0.84	0.12	< 0.01
20	3.94	0.20	0.01
22	10.70	0.35	0.01
24	32.32	0.72	0.01
26	87.80	0.51	0.01
28	943.23	0.86	0.01
30	9939.75	0.87	0.01
40	—	13.34	0.02
50	—	200.38	0.12
60	—	2882.23	0.06
70	—	10264.85	0.08
80	—	—	0.12
90	—	—	0.16
100	—	—	0.22

Note. ILP = Integer linear programming. Running time is given in seconds.

the simulation, the above mentioned methods were applied to the data set that was generated in this run, with the following exceptions: Because integer linear programming potentially has exponential running time but a lot of simulation runs were necessary for a meaningful evaluation, the exact integer linear programming technique was only applied to data sets for $N \leq 20$. Integer linear programming combined with the preclustering technique was applied to data sets for $N \leq 40$. Matching was only applied for $K = 2$ because the matching procedure is based on the selection of item pairs and cannot readily be extended to $K > 2$. The remaining methods were applied to all data sets.

We used the following evaluation criteria: First, we computed the anticluster editing objective D_{within} based on the Euclidean distance. The absolute values D_{within} were converted to percentages ($\%D_{\text{within}}$) using the following rule: First, the maximum value of D_{within} attained in a given run was determined. Next, the objective value attained by each method was divided by the maximum and multiplied by 100, ensuring that best value per run was 100%. Whenever the exact integer linear programming method was applied (i.e., when $N \leq 20$), $\%D_{\text{within}}$ could be interpreted as the percentage of the optimal value that was possible. For $N > 20$, $\%D_{\text{within}} = 100\%$ represented the best solution that was found in a run, that was however not necessarily optimal.

The anticluster editing objective was primarily computed to compare the performance of the different algorithms, in particular to evaluate how well the heuristic exchange method performs in comparison to the exact integer linear programming approaches. To compare the performance of the different partitioning methods (anticluster editing, k-means anticlustering, random assignment, matching), we additionally quantified how strongly the K sets differed in the features' means and standard deviations. To this end, we computed the mean and standard deviation on each feature for each anticluster. For each feature, we then computed the absolute difference between the minimum and maximum value of the means and standard deviations between the anticlusters. The mean of the absolute differences across all

features (where the number of features could vary between 1 and 4) quantified the total dissimilarity in means (ΔM) and standard deviations (ΔSD). A lower value of ΔM and ΔSD indicates that the sets were more similar with regard to the respective criterion.

Table 2 shows the results of the simulation study. Anticluster editing (ACE) performed better than random assignment and matching on all criteria and across all sample sizes. Matching and random assignment showed even decreased performance when the stimulus pool was small. When N increased, the similarity of the different sets generally increased irrespective of the partitioning method. This result is evident from a general increased similarity in feature means and standard deviations when N was larger. Table 2 also illustrates that k-means anticlustering was most successful at minimizing differences in feature means. However, this came at the cost of neglecting the features' standard deviations. Even a simple random assignment consistently resulted in more similar standard deviations than k-means anticlustering that tended to over-optimize similarity in means. Anticluster editing accomplished a better tradeoff between minimizing differences in means as well as standard deviations.

The three anticluster editing algorithms performed similarly well. In comparison to the exact integer linear programming technique, preclustering hardly impaired the solution quality. Moreover, the average objective D_{within} found by the heuristic exchange algorithm was always within a small margin ($\leq 0.3\%$) of the exact approaches. This constitutes a remarkable performance for a heuristic that does not make any guarantees with regard to optimality. Therefore, the `anticlust` program includes useful optimization algorithms for small to large stimulus pools.

Applications

We present three applications to illustrate how the `R` package `anticlust` can be used to partition data sets into similar parts. The package `anticlust` is freely available from the

Table 2

Results of the simulation study.

	$K = 2$			$K = 3$		
	$\%D_{\text{within}}$	ΔM	ΔSD	$\%D_{\text{within}}$	ΔM	ΔSD
$\boldsymbol{N} \in \{10, ..., 20\}$						
ACE-ILP	100.00	0.13	0.24	100.00	0.27	0.50
ACE-ILP/Preclustering	99.99	0.14	0.22	99.95	0.27	0.50
ACE-Exchange	99.85	0.14	0.24	99.70	0.29	0.51
Matching	98.53	0.32	0.27	—	—	—
K-Means-Anticlustering	98.49	0.06	0.37	97.93	0.16	0.69
Random assignment	94.99	0.46	0.34	90.56	0.81	0.63
$\boldsymbol{N} \in \{21, ..., 40\}$						
ACE-ILP/Preclustering	100.00	0.05	0.11	100.00	0.12	0.25
ACE-Exchange	99.92	0.06	0.12	99.83	0.13	0.27
Matching	99.46	0.17	0.15	—	—	—
K-Means-Anticlustering	99.02	0.02	0.25	98.22	0.05	0.45
Random assignment	97.26	0.32	0.22	94.85	0.58	0.43
$\boldsymbol{N} \in \{42, ..., 100\}$						
ACE-Exchange	100.00	0.02	0.06	100.00	0.05	0.12
Matching	99.84	0.09	0.09	—	—	—
K-Means-Anticlustering	99.47	0.00	0.16	99.05	0.01	0.28
Random assignment	98.74	0.22	0.15	97.52	0.39	0.27

Note. ACE = Anticluster Editing. ILP = Integer linear programming. $\%D_{\text{within}}$ is the percentage of the best anticluster editing objective that was found in a simulation run. ΔM and ΔSD quantify how strongly means and standard deviations differed between anticlusters. Table cells contain the average objectives across simulation runs, split by N.

website https://github.com/m-Py/anticlust where installation instructions are included. All applications presented here can be reproduced completely by code that is provided in the accompanying OSF repository (Papenberg & Klau, 2019).

Application I: Stimulus assignment

Our first example makes use of a data set describing 96 stimuli that was courteously provided by Marie Luisa Schaper (Schaper et al., 2019a, 2019b). The data set is included in the package `anticlust` and available after installation. It can be accessed using the following R code:

```
library(anticlust) # load the package
data(schaper2019) # load the data set
```

The item pool consists of 96 words. Each word represents an object that is either typically found in a bathroom or in a kitchen (see Table 3). For their experiments, Schaper et al. manually partitioned the 96 words into 3 lists that should be as similar as possible with regard to four numeric criteria. Such manual partitioning is tedious work, usually consisting of numerous iterations of switching stimuli between sets, either until the sets are deemed to be similar enough or until the researcher runs out of patience. Despite the high effort, the results are usually suboptimal. In particular, it is difficult to manually minimize differences with regard to multiple numeric variables. As our example illustrates, anticlustering may save hours of daunting work and the results are strongly improved as compared to a manual assignment.

To partition the stimulus pool into three equal-sized sets, we make use of the main function of the package `anticlust`: `anticlustering()`. Its input is a data table where each row is a stimulus and each column is a numeric feature (that is, columns `T`, `A`, `S` and `F` in Table 3). A parameter `K` specifies the number of stimulus sets. Moreover, we employ the argument `categories` to balance the frequency of kitchen and bathroom items across sets.

Table 3

A subset of the item pool used by Schaper et al.

Object	Room	T	A	S	F
moisture mask	bathroom	4.10	1.04	5	21
sanitary pad	bathroom	4.22	1.12	4	19
hairspray	bathroom	4.32	1.13	2	17
tampon	bathroom	4.35	1.22	2	17
kitchen chair	kitchen	4.30	1.16	3	19
potato peeler	kitchen	4.32	1.00	3	18
sugar basin	kitchen	4.40	1.08	4	20
salad cutlery	kitchen	4.48	1.02	4	20
...	...	...	...	...	...

Note. Room = The room in which the object is typically found. T = Typicality: How expected is it to find the object in the typical room? A = Atypicality: How expected is it to find the object in the atypical room? S = number of syllables in the German object name. F = word frequency. Anticlustering was used to create three word lists that have minimal differences with regard to the four numeric variables while balancing out room frequency across lists.

This is consistent with the manual assignment by Schaper et al. who sought to make the experimental sets similar with regard to the numeric variables as well as to the categorical variable `Room`. Categorical restrictions are implemented through an adjustment of the exchange algorithm that `anticlustering()` calls by default: The initial assignment is conducted in such a way that the categories are balanced across anticlusters. Then, only items belonging to the same category are considered to be feasible exchange partners, ensuring that the categories remain balanced throughout. The following code first selects the columns containing the relevant numeric features, then calls the anticlustering procedure, and finally stores the output in a variable called `anticlusters`:

```
features <- schaper2019[, 3:6]
anticlusters <- anticlustering(
  features,
  K = 3,
  objective = "distance",
  method = "exchange",
  preclustering = FALSE,
  categories = schaper2019$room
)
```

The output of the `anticlustering()` function is a vector of length 96 where each entry is a number between 1 and 3 representing the anticluster each item has been assigned to. Note that the output can vary between different function calls because the initial state of the exchange algorithm is random. While the exchange algorithm is executed quickly with a pool of 96 items ($<$ 1s), the exact integer linear programming method is no longer applicable. With fewer items, the argument `method` could be set to `"ilp"` to obtain an optimal solution. Setting the optional argument `preclustering` to `TRUE` would additionally activate preclustering restrictions.

Table 4

Using anticlustering on the data set provided by Schaper et al.

	Typicality	Atypicality	Syllables	Frequency
Anticluster Editing				
List 1	4.49 (0.26)	1.10 (0.07)	3.44 (1.01)	18.28 (2.40)
List 2	4.49 (0.27)	1.11 (0.07)	3.38 (0.87)	18.31 (2.44)
List 3	4.50 (0.23)	1.11 (0.06)	3.44 (0.91)	18.34 (2.39)
ΔM (ΔSD)	0.01 (0.04)	0.01 (0.01)	0.06 (0.14)	0.06 (0.05)
K-Means Anticlustering				
List 1	4.49 (0.24)	1.10 (0.06)	3.41 (0.61)	18.31 (2.42)
List 2	4.49 (0.26)	1.10 (0.07)	3.41 (1.01)	18.31 (2.69)
List 3	4.49 (0.25)	1.10 (0.07)	3.44 (1.11)	18.31 (2.09)
ΔM (ΔSD)	0.00 (0.02)	0.00 (0.01)	0.03 (0.50)	0.00 (0.60)
Manual assignment				
List 1	4.49 (0.25)	1.10 (0.07)	3.31 (0.90)	17.94 (2.47)
List 2	4.49 (0.25)	1.10 (0.06)	3.41 (0.71)	18.56 (2.53)
List 3	4.49 (0.25)	1.10 (0.07)	3.53 (1.14)	18.44 (2.17)
ΔM (ΔSD)	0.00 (0.00)	0.00 (0.01)	0.22 (0.43)	0.62 (0.36)
Random assignment				
List 1	4.47 (0.26)	1.10 (0.06)	3.69 (0.78)	18.44 (2.08)
List 2	4.55 (0.25)	1.10 (0.07)	3.09 (1.06)	17.94 (2.54)
List 3	4.46 (0.24)	1.10 (0.07)	3.47 (0.84)	18.56 (2.54)
ΔM (ΔSD)	0.09 (0.02)	0.00 (0.01)	0.60 (0.28)	0.62 (0.46)

Note. Word lists were created via anticlustering, random assignment or manually by Schaper et al. (2019a, 2019b). Table cells contain means and standard deviations (in brackets) by word list. ΔM and ΔSD are the absolute difference between the minimum and maximum of the mean and standard deviation, respectively.

We applied both anticluster editing and k-means anticlustering to the data set by varying the parameter `objective` between the default option `"distance"` (anticluster editing) and `"variance"` (k-means anticlustering). When comparing the results of anticluster editing and k-means anticlustering, a typical picture is illustrated in Table 4: K-means anticlustering is best suited to optimize the similarity of the feature means; in all but one case, they perfectly agree on two decimals. Anticluster editing is better suited to minimize differences in standard deviations while also minimizing differences in means rather successfully. As also shown in Table 4, the manual assignment established by Schaper et al. successfully minimized differences in mean typicality ratings. It was, however, apparently difficult to minimize differences in word frequency and the number of syllables at the same time. This is a typical result; it is very difficult for people to equalize more than a few variables when creating similar lists. In contrast, anticlustering optimized similarity with regard to all four variables. Table 4 also illustrates the results of an entirely random stimulus assignment. The means and standard deviations differ most strongly in this case. Clearly, random assignment does not achieve the goal of creating similar stimulus sets.

Application II: Independent samples for cross validation

With the increasing availability of large data sets in psychological research, machine learning has become an increasingly important tool in data analysis (e.g., Chen & Wojcik, 2016; Adjerid & Kelley, 2018). In contrast to most classical methodology in psychological research, machine learning has a stronger focus on predicting behaviour rather than its explanation (Yarkoni & Westfall, 2017). The shift in focus towards prediction is accompanied with a stronger reliance on cross validation techniques to test the predictive performance of statistical models on independent data. By validating a model on independent data, cross validation prevents overfitting and allows for an unbiased estimate of its predictive accuracy (Chen & Wojcik, 2016). Because truly independent data sets are usually not available, cross validation instead relies on splitting the data set at hand into

multiple parts, applying a statistical model to one of them, and finally testing its predictive accuracy on the other parts (Koul, Becchio, & Cavallo, 2018). In the most simple case, the data is just split into two sets: a training set and a test set. The training set is used to build a statistical model; the test set is used for model validation. Another popular cross validation technique is k-fold cross validation where the data set is split into k parts (e.g., into 5 or 10 *folds*). The model is trained on $k - 1$ parts and validated on the remaining one; this process is repeated until each fold has been used as the validation sample.

In regular cross-validation, data sets are randomly split. A potential problem with random splitting is that it may cause an uneven distribution of the input variables across splits. This distortion may unwantedly affect the estimated prediction accuracy (Zeng & Martinez, 2000). To overcome this potential shortcoming, we illustrate how to use the `anticlust` package to partition a data sets into independent—but equivalent—samples for cross validation. We make use of a data set consisting of responses to the Narcissistic Personality Inventory (NPI; Raskin & Terry, 1988). The NPI consists of 40 items where each item is a forced choice between two statements. One of the statements is interpreted as a narcissistic self description. For example, one NPI item reads:

1. I am no better or worse than most people.
2. I think I am a special person.

For this item, statement 2 is interpreted as the narcissistic response. Choices are scored binary: 1 indicates that a narcissistic response was selected, 0 indicates that the narcissistic response was not selected. The NPI data set that we use in our example was openly published by the Open-Source Psychometrics Project and is freely available online.[3] After removing cases that include missing responses, the data set consisted of 10,440 cases, stored in the `R` variable `npi`. Our accompanying OSF repository contains the required code

[3] https://openpsychometrics.org/_rawdata/

to read the data, score the responses, and to select the relevant columns—this preprocessing has been left here to focus on the anticlustering application (Papenberg & Klau, 2019). We used k-means anticlustering to split the sample into five parts that could be used in a 5-fold cross validation. A function was employed that is specialized on processing large data sets (`fast_anticlustering()`), the details of which are discussed below. The code ran in about 4 minutes on a personal computer (Intel Core i7-7700, 3.60GHz).

```
samples <- fast_anticlustering(
  subset(npi, select = score_Q1:score_Q40),
  K = 5,
  categories = npi$gender,
  k_neighbours = 5
)
```

As first argument, we passed the binary item scores—a 10,440 × 40 response matrix—to ensure that responses to all 40 items were distributed similarly across the five folds. Additionally, we balanced participant gender across the five folds using the argument `categories = npi$gender`. This was done to illustrate how `anticlust` can conduct a stratified split for cross validation; it was not done to suggest that balancing gender is generally useful in cross validation.

As the result of anticlustering, the average test score (i.e., the average sum of the item scores) across the 40 items was 13.37—in each of the five cross validation samples. In comparison, an entirely random split resulted in average test scores of 13.12, 13.20, 13.28, 13.54, and 13.72. Even more impressively, since we applied anticlustering to individual items rather than aggregate test scores, item-level responses were also astonishingly similar across folds. For 38 out of the 40 items (95%), the mean item difficulties perfectly agreed on 2 decimals—across all five samples. With the random split, this never happened. Moreover, anticlustering also ensured that participant gender was balanced across folds (see Table 5),

Table 5

Frequency of self-reported gender across cross validation samples after using anticlustering.

	no gender chosen	male	female	other
Sample 1	2	1199	880	7
Sample 2	2	1199	879	8
Sample 3	2	1199	879	8
Sample 4	3	1199	879	7
Sample 5	3	1198	880	7

which is not at all taken into account when using a random split.

In this application, we used an `anticlust` function that was specifically implemented to work with large data sets: `fast_anticlustering()`. This function always employs k-means anticlustering because k-means is preferred over anticluster editing when N is large. The anticluster editing objective is given as the sum of pairwise dissimilarities, and therefore a quadratic inter-item dissimilarity matrix needs to be computed and stored in memory. K-means only requires to compute distances between each data point and its cluster center. In the current example, this results in holding 54,491,580 (anticluster editing) versus 52,200 (k-means anticlustering) distances. To reduce the computational burden further, `fast_anticlustering()` has an argument `k_neighbours` to determine the number of exchange partners per element. By default, exchanges are attempted with all persons who are currently assigned to a different anticluster and who are of the same gender. This implies that several thousand exchanges are realized for each of the 10,440 persons, which is not feasible in acceptable running time. When using `k_neighbours = 5`, only 5 exchanges realized for each person, reducing the computational burden considerably. Who serves as exchange partner is determined using a nearest neighbour search that is implemented

through the `R` package `RANN` (Arya, Mount, Kemp, & Jefferis, 2019; Arya & Mount, 1993). That is, the five most similar persons of the same gender served as exchange partners. Using nearest neighbours follows the same logic as preclustering: by swapping only very similar data points between anticlusters, only exchanges are conducted that are consistent with a high between-cluster similarity.

Application III: Parallel test splits

In educational psychology, it is often of interest to split an item pool into parts of equal difficulty (Gierl et al., 2017), and ideally of equal item discrimination (Brusco et al., 2013). The most important example is the creation of parallel versions of an examination to make students' achievement comparable between different cohorts. Anticlustering can be used to achieve this goal. For the purpose of illustration, we used the NPI data set from the previous example to split the 40 items into four equivalent parts. First, we computed item difficulty and part-whole corrected item discrimination indices for all 40 items; item indices were stored in a two-column matrix (variable `item_indices` in `R`). The code used for this preprocessing can be retrieved from the accompanying OSF repository (Papenberg & Klau, 2019). After preprocessing, we applied anticluster editing as follows:

```
split <- anticlustering(
  item_indices,
  K = 4
)
```

The code was executed in less than a second. The resulting descriptive statistics are shown in Table 6. Anticluster editing minimized differences with regard to both item difficulty and item discrimination between test sets, showing that anticlustering can successfully be used to create fair parallel tests in the context of school or university examinations. Again, a random split performed far worse than anticlustering (see Table 6).

Table 6

Descriptive item statistics by item set after random assignment and anticlustering.

	Item difficulty	Item discrimination
Anticluster Editing		
Set 1	0.33 (0.11)	0.42 (0.10)
Set 2	0.33 (0.12)	0.42 (0.09)
Set 3	0.34 (0.14)	0.43 (0.07)
Set 4	0.33 (0.13)	0.43 (0.08)
Random assignment		
Set 1	0.34 (0.10)	0.45 (0.08)
Set 2	0.32 (0.14)	0.42 (0.08)
Set 3	0.37 (0.14)	0.39 (0.09)
Set 4	0.30 (0.10)	0.45 (0.08)

Note. Table cells contain means and standard deviations (in brackets).

Discussion

We presented anticlustering as a method to partition a pool of elements into equivalent parts, a task that is ubiquitous in psychological research. Even though the usefulness of anticlustering has been exploited across a variety of research fields, it has only recently been recognized as a potential tool for researchers in psychology (Brusco et al., in press). By contributing our free and open source `R` package `anticlust`, researchers are now able to apply the anticlustering methodology easily.

We described two anticlustering methods: k-means anticlustering and anticluster editing, the latter also known as the maximum diverse grouping problem (Fan et al., 2011).

K-means anticlustering is most successful at minimizing differences in cluster means, but may neglect similarity of standard deviations. For some applications such as creating parallel stimulus sets, it may not be desirable just to focus on the mean (e.g., van Casteren & Davis, 2007; Smith & Kutas, 2015). It is well known that the mean may perform poorly as a measure of overall cluster similarity: similar means can be obtained even when the underlying distributions clearly differ (Anscombe, 1973). Hence, some caution is needed when applying k-means anticlustering to assemble similar sets. Anticluster editing achieves a better balance between minimizing differences in means and standard deviations and should be preferred whenever overall similarity—and not just average similarity—should be maximized. However, we generally advocate to inspect the quality of an anticlustering solution by comparing the descriptive statistics between sets, no matter what method has been employed. For large data sets, k-means anticlustering is more efficient and should be preferred.

In `anticlust`, we implemented an exchange method as our primary algorithmic tool. As confirmed in our simulation study, the exchange method is well suited to solve anticlustering problems; its performance was consistently close to the exact integer linear programming methods. However, on large data sets, the exchange algorithm may be outperformed by meta-heuristics that have recently received increasing interest (Gallego et al., 2013; Palubeckis et al., 2015). Nevertheless, we opted to implement the exchange algorithm because it was conveniently customizable to allow for categorical restrictions, i.e., balancing nominal variables between groups. Categorical restrictions have usually not been taken into consideration in anticlustering algorithms, but they are crucial for applications in psychology. Additionally, by adjusting the number of exchange partners, we were able to successfully apply anticlustering to large problem instances and the results were excellent (see Application II).

To conclude, anticlustering is a—now accessible—tool that quickly partitions an item pool into equivalent parts, therefore solving a problem that frequently occurs in

psychological research, but so far lacked an appropriate software solution. Anticlustering can be applied easily using the free and open source R package `anticlust`. Based on the demands of the user community, we intend to maintain and further improve this software package in the future.

References

Adjerid, I., & Kelley, K. (2018). Big data in psychology: A framework for research advancement. *American Psychologist*, *73*(7), 899–917.

Aloise, D., Deshpande, A., Hansen, P., & Popat, P. (2009). NP-hardness of euclidean sum-of-squares clustering. *Machine Learning*, *75*(2), 245–248.

Anscombe, F. J. (1973). Graphs in statistical analysis. *The American Statistician*, *27*(1), 17–21. https://doi.org/10.1080/00031305.1973.10478966

Armstrong, B. C., Watson, C. E., & Plaut, D. C. (2012). SOS! An algorithm and software for the stochastic optimization of stimuli. *Behavior Research Methods*, *44*, 675–705. https://doi.org/10.3758/s13428-011-0182-9

Arya, S., Mount, D., Kemp, S. E., & Jefferis, G. (2019). *RANN: Fast nearest neighbour search (wraps ANN library) using L2 metric*. Retrieved from https://CRAN.R-project.org/package=RANN

Arya, S., & Mount, D. M. (1993). Approximate nearest neighbor queries in fixed dimensions. In V. Ramachandran (Ed.), *Proceedings of the fourth annual ACM-SIAM symposium on discrete algorithms* (pp. 271–280). Philadelphia, PA: Society for Industrial; Applied Mathematics.

Aust, F., & Barth, M. (2018). papaja: Create APA manuscripts with R Markdown. Retrieved from https://github.com/crsh/papaja

Baker, K. R., & Powell, S. G. (2002). Methods for assigning students to groups: A study of alternative objective functions. *Journal of the Operational Research Society*, *53*(4), 397–404.

Bansal, N., Blum, A., & Chawla, S. (2004). Correlation clustering. *Machine Learning*, *56*,

89–113. https://doi.org/10.1023/B:MACH.0000033116.57574.95

Böcker, S., Briesemeister, S., & Klau, G. W. (2011). Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, *60*, 316–334. https://doi.org/10.1007/s00453-009-9339-7

Bravo, H. C., & Theussl, S. (2016). Rcplex: R interface to CPLEX. Retrieved from https://CRAN.R-project.org/package=Rcplex

Brusco, M. J., Cradit, J. D., & Steinley, D. (in press). Combining diversity and dispersion criteria for anticlustering: A bicriterion approach. *British Journal of Mathematical and Statistical Psychology*. https://doi.org/10.1111/bmsp.12186

Brusco, M. J., & Köhn, H.-F. (2009). Clustering qualitative data based on binary equivalence relations: Neighborhood search heuristics for the clique partitioning problem. *Psychometrika*, *74*(4), 685–703.

Brusco, M. J., Köhn, H. F., & Steinley, D. (2013). Exact and approximate methods for a one-dimensional minimax bin-packing problem. *Annals of Operations Research*, *206*(1), 611–626.

Brusco, M. J., & Stahl, S. (2001). Compact integer-programming models for extracting subsets of stimuli from confusion matrices. *Psychometrika*, *66*(3), 405–419.

Brusco, M. J., & Steinley, D. (2006). Clustering, seriation, and subset extraction of confusion data. *Psychological Methods*, *11*(3), 271–286.

Brusco, M. J., & Steinley, D. (2010). K-balance partitioning: An exact method with applications to generalized structural balance and other psychological contexts. *Psychological Methods*, *15*(2), 145–157.

Buchner, A., & Wippich, W. (2000). On the reliability of implicit and explicit memory

measures. *Cognitive Psychology*, *40*, 227–259. https://doi.org/10.1006/cogp.1999.0731

Chen, E. E., & Wojcik, S. P. (2016). A practical guide to big data research in psychology. *Psychological Methods*, *21*(4), 458–474.

Chen, P.-H. (2017). Should We Stop Developing Heuristics and Only Rely on Mixed Integer Programming Solvers in Automated Test Assembly? A Rejoinder to van der Linden and Li (2016). *Applied Psychological Measurement*, *41*, 227–240. https://doi.org/10.1177/0146621617695523

Cutler, A. (1981). Making up materials is a confounded nuisance, or: Will we able to run any psycholinguistic experiments at all in 1990? *Cognition*, *10*, 65–70. https://doi.org/10.1016/0010-0277(81)90026-3

Desrosiers, J., Mladenović, N., & Villeneuve, D. (2005). Design of balanced MBA student teams. *Journal of the Operational Research Society*, *56*(1), 60–66.

Dry, M. J., & Storms, G. (2009). Similar but not the same: A comparison of the utility of directly rated and feature-based similarity measures for generating spatial models of conceptual data. *Behavior Research Methods*, *41*, 889–900. https://doi.org/10.3758/BRM.41.3.889

Fan, Z., Chen, Y., Ma, J., & Zeng, S. (2011). A hybrid genetic algorithmic approach to the maximally diverse grouping problem. *Journal of the Operational Research Society*, *62*(7), 1423–1430.

Feo, T. A., & Khellaf, M. (1990). A class of bounded approximation algorithms for graph partitioning. *Networks*, *20*(2), 181–195.

Gallego, M., Laguna, M., Marti, R., & Duarte, A. (2013). Tabu search with strategic oscillation for the maximally diverse grouping problem. *Journal of the Operational*

Research Society, *64*(5), 724–734.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: Freeman.

Gierl, M., Daniels, L., & Zhang, X. (2017). Creating parallel forms to support on-demand testing for undergraduate students in psychology. *Journal of Measurement and Evaluation in Education and Psychology*, *8*(3), 288–302.

Glover, F., Kuo, C.-C., & Dhir, K. S. (1998). Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, *19*(1), 109–132.

Grötschel, M., & Wakabayashi, Y. (1989). A cutting plane algorithm for a clustering problem. *Mathematical Programming*, *45*, 59–96. https://doi.org/10.1007/BF01589097

Heck, D. W., & Erdfelder, E. (2016). Extending multinomial processing tree models to measure the relative speed of cognitive processes. *Psychonomic Bulletin & Review*, *23*, 1440–1465. https://doi.org/10.3758/s13423-016-1025-6

IBM. (2019). ILOG CPLEX Optimization studio [Integer linear programming solver]. Retrieved from https://www.ibm.com/products/ilog-cplex-optimization-studio

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*(8), 651–666.

Karp, R. M. (1986). Combinatorics, complexity, and randomness. *Communications of the ACM*, *29*(2), 97–109.

Koul, A., Becchio, C., & Cavallo, A. (2018). Cross-validation approaches for replicability in psychology. *Frontiers in Psychology*, *9*, 1117.

Krass, D., & Ovchinnikov, A. (2006). The university of toronto's rotman school of management uses management science to create mba study groups. *Interfaces*, *36*(2), 126–137.

Kroneisen, M., & Bell, R. (2018). Remembering the place with the tiger: Survival processing can enhance source memory. *Psychonomic Bulletin & Review*, *25*, 667–673. https://doi.org/10.3758/s13423-018-1431-z

Lahl, O., & Pietrowsky, R. (2006). EQUIWORD: A software application for the automatic creation of truly equivalent word lists. *Behavior Research Methods*, *38*, 146–152. https://doi.org/10.3758/BF03192760

Lahl, O., Wispel, C., Willigens, B., & Pietrowsky, R. (2008). An ultra short episode of sleep is sufficient to promote declarative memory performance. *Journal of Sleep Research*, *17*, 3–10. https://doi.org/10.1111/j.1365-2869.2008.00622.x

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. New York: John Wiley & Sons.

Nosofsky, R. M. (1992). Similarity scaling and cognitive process models. *Annual Review of Psychology*, *43*, 25–53. https://doi.org/10.1146/annurev.ps.43.020192.000325

Palubeckis, G., Ostreika, A., & Rubliauskas, D. (2015). Maximally diverse grouping: An iterated tabu search approach. *Journal of the Operational Research Society*, *66*(4), 579–592.

Papenberg, M. (2019). Anticlust: Subset partitioning via anticlustering. Retrieved from https://github.com/m-Py/anticlust

Papenberg, M., & Klau, G. W. (2019). Anticlustering [Open Science Framework Repository]. Retrieved from https://doi.org/10.17605/OSF.IO/CD5SR

Raskin, R., & Terry, H. (1988). A principal-components analysis of the narcissistic personality inventory and further evidence of its construct validity. *Journal of Personality and Social Psychology*, *54*(5), 8–902.

R Core Team. (2019). R: A language and environment for statistical computing. Retrieved from https://www.R-project.org/

Rokach, L., & Maimon, O. (2005). Clustering methods. In O. Maimon & L. Rokach (Eds.), *Data mining and knowledge discovery handbook* (pp. 321–352). New York: Springer.

Schaper, M. L., Kuhlmann, B. G., & Bayen, U. J. (2019a). Metacognitive expectancy effects in source monitoring: Beliefs, in-the-moment experiences, or both? *Journal of Memory and Language*, *107*, 95–110. https://doi.org/10.1016/j.jml.2019.03.009

Schaper, M. L., Kuhlmann, B. G., & Bayen, U. J. (2019b). Metamemory expectancy illusion and schema-consistent guessing in source monitoring. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *45*, 470–496. https://doi.org/10.1037/xlm0000602

Shamir, R., Sharan, R., & Tsur, D. (2004). Cluster graph modification problems. *Discrete Applied Mathematics*, *144*, 173–182. https://doi.org/10.1016/j.dam.2004.01.007

Smith, N. J., & Kutas, M. (2015). Regression-based estimation of ERP waveforms: II. Nonlinear effects, overlap correction, and practical considerations. *Psychophysiology*, *52*, 169–181. https://doi.org/10.1111/psyp.12320

Späth, H. (1986). Anticlustering: Maximizing the variance criterion. *Control and Cybernetics*, *15*(2), 213–218.

Steghöfer, J.-P., Behrmann, P., Anders, G., Siefert, F., & Reif, W. (2013). HiSPADA: Self-organising hierarchies for large-scale multi-agent systems. In M. Bauer, R.

Calinescu, M. Grottke, & B. Dillenseger (Eds.), *Proceedings of the IARIA international conference on autonomic and autonomous systems (ICAS)* (pp. 71–76). Lisbon, Portugal: International Academy, Research, and Industry Association.

Steinley, D. (2006). K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, *59*(1), 1–34.

Tversky, A. (1977). Features of similarity. *Psychological Review*, *84*, 327–352. https://doi.org/10.1037/0033-295X.84.4.327

Uroševic, D. (2016). Variable neighborhood search for maximum diverse grouping problem. *Yugoslav Journal of Operations Research*, *24*(1), 21–33.

Valev, V. (1983). Set partition principles. In J. Kozesnik (Ed.), *Transactions of the ninth Prague conference on information theory, statistical decision functions, and random processes (Prague, 1982)* (pp. 251–256). Prague: Springer Netherlands.

Valev, V. (1998). Set partition principles revisited. In A. Amin, D. Dori, P. Pudil, & H. Freeman (Eds.), *Advances in pattern recognition. SSPR /SPR 1998. Lecture notes in computer science* (pp. 875–881). Heidelberg: Springer.

van Casteren, M., & Davis, M. H. (2007). Match: A program to assist in matching the conditions of factorial experiments. *Behavior Research Methods*, *39*, 973–978. https://doi.org/10.3758/BF03192992

van der Linden, W. J. (2005). *Linear models for optimal test design.* New York: Springer Science.

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science*, *32*, 939–984. https://doi.org/10.1080/03640210801897856

Weitz, R., & Lakshminarayanan, S. (1997). An empirical comparison of heuristic and graph

theoretic methods for creating maximally diverse groups, vlsi design, and exam scheduling. *Omega*, *25*(4), 473–482.

Weitz, R., & Lakshminarayanan, S. (1998). An empirical comparison of heuristic methods for creating maximally diverse groups. *Journal of the Operational Research Society*, *49*(6), 635–646.

Wickham, H., François, R., Henry, L., & Müller, K. (2019). Dplyr: A grammar of data manipulation. Retrieved from https://CRAN.R-project.org/package=dplyr

Wittkop, T., Emig, D., Lange, S., Rahmann, S., Albrecht, M., Morris, J. H., ... Baumbach, J. (2010). Partitioning biological data with transitivity clustering. *Nature Methods*, *7*, 419–420. https://doi.org/10.1038/nmeth0610-419

Yarkoni, T., & Westfall, J. (2017). Choosing prediction over explanation in psychology: Lessons from machine learning. *Perspectives on Psychological Science*, *12*(6), 1100–1122.

Zahn, C. T. (1964). Approximating symmetric relations by equivalence relations. *Journal of the Society for Industrial and Applied Mathematics*, *12*(4), 840–847.

Zeng, X., & Martinez, T. R. (2000). Distribution-balanced stratified cross-validation for accuracy estimation. *Journal of Experimental & Theoretical Artificial Intelligence*, *12*, 1–12. https://doi.org/10.1080/095281300146272