MethodsX 8 (2021) 101545

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

A new tool for equating lexical stimuli across experimental conditions ☆

Evan N. Lintz*, Phui Cheng Lim, Matthew R. Johnson

Department of Psychology, University of Nebraska-Lincoln, Lincoln, NE 68588, USA

A B S T R A C T

In cognitive psychology and psycholinguistics, lexical characteristics can drive large effects, which can create confounds when word stimuli are intended to be unrelated to the effect of interest. Thus, it is critical to control for these potential confounds. As an alternative to randomly assigning word bank items to stimulus lists, we present LIBRA (Lexical Item Balancing & Resampling Algorithm), a MATLAB-based toolbox for quickly generating stimulus lists of user-determined length and number that can be closely equated on any number of lexical properties. The toolbox comprises two scripts: a genetic algorithm that performs the inter-list balancing, and a tool for filtering/trimming long omnibus word lists based on simple criteria, prior to balancing. Relying on randomized procedures often results in substantially unbalanced experimental conditions, but our method guarantees that the lists used for each experimental condition contain no meaningful differences. Thus, the lexical characteristics of the specific words used will add an absolute minimum of bias/noise to the experiment in which they are applied.

- Our toolbox balances word lists for arbitrary lexical properties to control confounds in cognitive psychology research.
- Our toolbox performs more efficiently than pure randomization or balancing manually.
- A graphical user interface is provided for ease of use.

A R T I C L E I N F O

Method name: Lexical Item Balancing & Resampling Algorithm (LIBRA)
Keywords: Cognitive psychology, Lexical, Wordlist, Genetic algorithm, English Lexicon Project
Article history: Received 22 April 2021; Accepted 8 October 2021; Available online 11 October 2021

☆ This work was supported in part by NSF/EPSCoR grant 1632849 and by the National Institute of General Medical Sciences of the National Institutes of Health [grant number P20 GM130461] and the Rural Drug Addiction Research Center at the University of Nebraska-Lincoln. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or the University of Nebraska.

DOI of original article: 10.1016/j.cognition.2021.104655

* Corresponding author.
E-mail address: evan.lintz@huskers.unl.edu (E.N. Lintz).

https://doi.org/10.1016/j.mex.2021.101545

Specifications table

Subject area	Psychology
More specific subject area	*Cognitive Psychology*
Method name	Lexical Item Balancing & Resampling Algorithm (LIBRA)
Name and reference of original method	*N/A. This method improves upon the standard practice of using random assignment of stimuli to experimental conditions, thus there is no reference for the use of random assignment as it is traditionally employed.*
Resource availability	The software necessary to implement this method is open-source and can be downloaded from: https://www.mathworks.com/matlabcentral/fileexchange/79628-lexical-item-balancing-resampling-algorithm-libra

Introduction

In this section, we discuss how imbalances in lexical properties can create potential confounds in cognitive psychology and psycholinguistics studies, and detail some situations in which this may occur. We briefly describe several programs developed to address this issue, including our own presented in this paper (LIBRA: Lexical Item Balancing & Resampling Algorithm), in the hope that it will help researchers select the tool(s) best suited to their needs.

Concerns

An extensive amount of work in experimental psychology has been devoted to language, and while lexical stimuli are an integral part of psycholinguistics, their application extends broadly across the field of psychology. For example, in working memory (WM) paradigms, the lexicon is a natural corpus of varied, distinctive stimuli that affords straightforward item-level testing of memory. Similarly, when probing long-term memory (LTM), words are an especially useful and flexible stimulus category. For areas of psychology, especially cognitive psychology, that employ lexical stimuli, the specific words used are often not the subject of study, but rather a means to an end. In order to investigate phenomena that are, in principle, independent of the specific word stimuli used, we must consider how to properly control for item-level effects.

Extensive work in cognitive psychology and psycholinguistics has demonstrated that lexical characteristics are capable of driving large effects. Word length [2], usage frequency [13,15,16], and emotional valence [7] are just a few of the properties known to influence dependent measures in a variety of tasks and contexts. In psycholinguistic studies, the lexical properties themselves are often the independent variable, manipulated to demonstrate the effect of that property. However, when word lists serve a utility function and are intended to be unrelated to the effect of interest, such as in most WM or LTM paradigms, it is critical to control for these potentially confounding lexical effects, particularly because they can be considerably larger than the effect of actual interest. This was demonstrated by Balota and colleagues [4] using a lexical decision task (LDT), who reported that a relatively small set of lexical properties accounted for a considerable portion of response time (RT) variation ($R^2 = .42$).

Considerations

In a typical experiment scenario, different conditions require separate, disjoint lists of word stimuli. Consider a verbal WM experiment in which the RT of a button press to a lexical memory probe is the main dependent measure of a within-subjects manipulation with two levels. In this case, we do not wish to reuse words due to potential carryover effects, and so two lists of unique words are required. However, the time it takes to read each word will have a direct impact on RT. Therefore, if one stimulus list has a higher mean reading time than another's, that difference would carry over to RT analyses of the experimental manipulation. If not carefully controlled, this lexical property effect may manifest in a number of ways. For example, it could introduce systematic bias if each subject receives the same stimulus list, or, if word lists are randomized separately for each participant, the variance introduced by the lexical effects can represent a considerable source of noise compared to the size of the effect(s) of interest.

Some lexical properties are intrinsic, such as a word's length or number of syllables; however, other properties must be measured from human behavior. One of the latter properties is frequency of use, a property whose influence on performance is self-evident (frequently accessed words are recognized and produced more easily than rare ones; [3]) but which is elusive to measure definitively. This is partly due to difficulties in operationalizing the concept (e.g., do we consider written language, spoken, or both? How variable is word frequency between individuals or demographic/geographic groups?) and partly due to the ever-shifting nature of language use over time. Clearly, it is impractical for individual research teams to re-measure frequency and other behavior-derived properties with each new experiment, so we must rely on previously published values from large-scale lexicon studies. While older work relied on frequency measures derived from various books and texts (e.g., [13,16]), modern technology has allowed frequency measures to consider online sources (e.g., [6]) and has facilitated the compilation of large databases of word properties. A number of megastudies have now been published that provide ratings and behavioral norms for tens of thousands of words ([5,8]; for a review, see [12]). These are powerful tools for researchers using lexical stimuli, as one can easily select subsets of these databases based on the needs of a particular experiment. For example, an experimenter interested in the influence of a word's valence on its recall might require one list of positively valenced words and another of negatively valenced words.

While obtaining an initial word bank from such a database is an excellent starting point for selecting experimental stimuli based on a set of desired lexical properties, one must also consider the converse problem: eliminating the potential influence of undesired properties. Often, words from this initial bank must then be allocated to sub-lists for use with different conditions or experimental blocks, which is commonly done via randomization. As we demonstrate later, this use of random assignment, while common, is not guaranteed to equate lexical properties across lists; in fact, depending on the size and number of sub-lists that are created, it can lead to significantly unbalanced lists.

Software solutions

Balancing the word lists to ensure they are equated on key lexical properties can be done by hand if the number of lexical properties and sub-lists is very small, but this can be tedious and time-consuming. Because the number of possible permutations to check increases dramatically with the number of lexical properties, it becomes increasingly unworkable to do this for more than two properties or sub-lists. Certainly, we are not the first to apply an algorithmic approach to balancing stimulus lists; however, the tools commonly available may not be well suited to all applications. Furthermore, available tools do not achieve the goal of fully automating the process; some require manual item selection or trimming during the balancing process, and none that we are aware of are able to automatically trim omnibus word lists of potential confounds such as homophones or compound words. While most software tools can be applied to a variety of use cases, they are often tailored specifically to psycholinguistic applications (i.e., experiments wherein the lexical properties of the word stimuli are an independent variable to be manipulated). For experiments that treat words as simply another category of stimuli, not meant to be related to the effects of interest (e.g., memoranda in a working memory task), the available tools may prove to be, in many cases, either insufficient or overly complex. We will briefly review a few extant methods for list balancing and describe how they could be improved upon for meeting the needs of a typical cognitive psychology experiment (see below, Methods/Application, for an example of one such use case).

"Match" [17] applies a relatively simple pairwise matching algorithm that attempts to balance any number of lists, with list members having an optimal match with another member of all other lists, on all the relevant dimensions. For example, three lists of words might be generated in which items 1a, 2a, and 3a are matched for number of letters, number of syllables, and number of phonemes; pairwise matching continues among items 1b, 2b, 3b, etc. The match quality is quantified with a Euclidean distance metric that considers all the matched items' values for each dimension to be balanced. While this approach may lead to satisfactorily balanced lists, it is difficult for a user to tell how close the current best result is to a theoretically optimal solution, and the Euclidean distance metric becomes decreasingly interpretable the more dimensions the researcher attempts to balance. For

those who require lists to be balanced overall (as opposed to just pairwise), the matching approach may even introduce confounds. For example, the closest matches identified pairwise may lead to small differences between items (e.g., six letter words matched with seven letter words); however, if the one-letter differences become systematic, and one list contains shorter words on average, the overall list differences may be significant. Note that Match users must be minimally proficient with command-line scripting, as the software lacks a graphical user interface (GUI).

Rather than develop a new software tool, Guasch and colleagues [9] describe a method by which one can apply cluster analyses within the commercial statistical software SPSS to inform list balancing procedures. Cluster analysis seeks to divide an overall pool of items into *k* groups that are as similar to other group members as possible on the relevant lexical dimensions, while also maximizing inter-group differences. The authors describe a heuristic to estimate a value for *k* that should generate clusters of sufficient size to populate lists of items such that each cluster contributes items to each experimental condition's list equally, thus in principle balancing the lists. Functionally, this approach is similar to the pairwise matching described above and also uses a Euclidean distance metric; as such, it suffers the same drawbacks when applied to our typical use case. The *k*-means approach also requires a fair amount of researcher involvement with the clustering choices, analysis, and other aspects of the method, and as such may consume more of the researcher's time than a more automated process.

Armstrong and colleagues [1] developed a comprehensive software package built upon MATLAB (MathWorks, Natick, MA) to address a wide variety of list balancing needs. Their method, Stochastic Optimization of Stimuli (SOS), applies an algorithm to search for potential balancing solutions that satisfy any number of researcher-imposed constraints (e.g., minimizing differences between lists on some dimensions while maximizing differences between them on other dimensions). SOS provides a variety of ways to quantify cost (e.g., Euclidean distance, which is used in the previous two examples; entropy; correlation), allowing researchers to tailor balancing procedures to the needs of a specific experiment. The large amount of customization available gives SOS an advantage over Match and *k*-means clustering in terms of achieving near-optimal solutions and fitting a wide variety of experimenters' needs; however, the cost of such flexibility and power is dramatically increased complexity, which may be daunting for a typical user. While there is GUI functionality, setting up a balancing task with even a handful of constraints requires considerable work within the GUI; scripting the steps with MATLAB code can streamline the process, but of course this requires relatively advanced knowledge of the MATLAB programming language and potentially a fairly close reading of the SOS source code. Thus, as mentioned, SOS is easily the most flexible and powerful tool on this list, and would be well suited for users with specific and complex sets of constraints, but it also requires a great deal of MATLAB proficiency and an intensive workflow, and thus may not be an ideal choice for users with more modest needs and/or more limited technical expertise.

Additionally, all of the software solutions described above still leave one rather large task to manual processes. When compiling lists of word stimuli, one common approach can be to obtain an initial pool of words from a massive database, and then cut those lists down according to the needs of the project. While these databases may allow some degree of initial selection constraints to be imposed (e.g., word length, word frequency), the lists obtained may still contain confounds and undesirable items. Researchers must then manually trim items such as homophones, proper nouns, compound words, or variants of a word (e.g., abandon/abandoned/abandoning; apple/apples); a potentially time-consuming task, when word pools may contain tens of thousands of items.

As an additional alternative to the previously existing software options, we present here LIBRA (Lexical Item Balancing & Resampling Algorithm), a software tool for quickly filtering word pools (e.g., removal of homophones) and generating stimulus word lists of user-determined length and number, which can be closely equated on an arbitrary number of lexical properties. Compared to the above options, LIBRA offers similar balancing functionality to the Match and *k*-means approaches, but with a GUI that requires no programming or command-line proficiency, and a significantly simpler and more user-friendly interface than SOS (at the expense of some flexibility), while also adding the aforementioned filtering functionality that is not, to our knowledge, currently offered by any similar software tools.

Method

In this section, we describe our approach to generating stimulus lists using a genetic algorithm (Approach), provide a step-by-step explanation of how to use LIBRA and its graphical user interface (Graphical user interface and usage), and demonstrate that LIBRA both runs quickly and generates well-balanced stimulus lists (Validation). Those interested in the logic behind the algorithm should start from the "Approach" section; those who want to skip straight to using the tool should read the "Genetic Algorithm Implementation" section if they wish to understand the tool's mechanics, and then focus on the three sections "Graphical user interface (GUI) and usage," "Balancing script," and "Filtering script." Note that LIBRA consists of two MATLAB scripts, one for balancing and one for filtering word banks; filtering should be run before balancing, although researchers with curated word banks may wish to skip this step.

Approach: Optimizing with a genetic algorithm

Our approach to equating word lists uses a basic genetic algorithm (GA) to ensure that the final stimulus lists for each experimental condition are balanced as closely as possible across lexical properties that could spuriously influence our dependent measure. This method has been successfully applied in several previous studies [10,11,14] and additional functionality has been incorporated for the software toolbox described in this paper. The GA is a class of optimization algorithm loosely based on evolutionary theory, and thus it is often described with terms rooted in this analogy. The parameters or values we are attempting to optimize act as "genes" that collectively make up individual members of a population of "organisms" where each organism is a potential solution. Some of these individuals (sets of values) will be more fit (optimal) than others. GAs are useful for problems: (1) that have many possible configurations of parameters (too many for an exhaustive search), (2) that do not require the absolute maximum fit to be found (merely a very good one), and (3) in which small changes to the "genome" typically produce similarly small changes in fitness. In principle, one could achieve the same goal more simply by generating a large number of randomized solutions until an adequate one is found. However, this process would be too inefficient in many cases, since adequate solutions might represent a vanishingly small percentage of the distribution of possible solutions. Furthermore, we may frequently come across a solution that is *nearly* optimal but gets discarded since it does not meet our requirements. GAs allow us to capitalize on the near-misses by making subsequent candidate solutions that are slight "mutations" of the best candidates of previous rounds, rather than full re-randomizations. Such "child" solutions are more likely to outperform their "parents" than a totally randomized genome would be. The process is repeated, iteratively producing new child solutions until one is found that meets our target fitness score. Thus, as in nature, the most fit individuals are selected to pass on their traits, thereby ensuring an incremental improvement in fitness with each future generation. Notably, a solution found in this manner might also have been found using a pure random-sampling approach; however, in problem domains well-suited to GAs, we are likely to find it much faster using evolutionary methods than by continually re-rolling the dice hoping to stumble on a suitable solution.

For optimizing word lists specifically, the lexical properties that we wish to balance represent the "genes" of the words used, that in turn are the "organisms" of our population (word lists). Population mutations are achieved by swapping individual words out of the parent word lists until a child is created whose individuals are more closely equated (optimal fitness). There are numerous variations of the basic GA, with alternative methods of addressing factors such as population size or the rate and method of mutation, to name a few. Here we have implemented a fairly simple version of a GA, as these more complex versions are unnecessary to achieve our target fitness goals.

Application

In Lintz and Johnson [14], we sought to explore the putative WM processes of refreshing and removal. Participants were asked to perform a lexical decision task (LDT) as well as a surprise LTM test. Thus, we were concerned with controlling the lexical properties that might be expected to influence our main dependent measures; LDT reaction time in the first task, and confidence ratings

of memory strength in the second. We identified five such properties: number of letters, number of phonemes, number of syllables, frequency of use, and the average time to read aloud. As an accuracy check in the LTM test, foil words (not previously seen) were interspersed with the words that had previously been seen in the LDT. Thus, it was important to not only ensure balancing between the three lists of words presented in the LDT, but also to equate the foil words presented in the LTM task with each other list.

Although it is impossible to exactly equate four lists on five lexical properties each down to the last decimal point, we want them to be as closely equated as possible, to minimize their chances of spuriously influencing our results. It is not uncommon for studies like this to report that their lists were not significantly different from each other (i.e., $p > .05$ for all of a series of two-sample t-tests). However, "non-significantly" different does not mean "no difference"; even if the differences are not large enough to meet the conventional (and arbitrary) threshold of statistical significance, their magnitude may still be sufficient to add a fair bit of noise or bias to the results. This is especially true if the differences go in directions that may reinforce each other; for example, if List A has fewer mean syllables than List B ($p = .07$) and higher mean frequency ($p = .06$), both differences that would be associated with faster RTs to List A, this may present more of a problem for bias than if List A had fewer syllables ($p = .04$) and *lower* frequency ($p = .03$), as the latter pair of differences would tend to cancel each other's RT effects out. So, ideally we would want to go beyond mere non-significance when equating lists; but in order to apply a GA to this balancing problem, we require a fitness function by which to evaluate the equality of lists, and for this purpose the *p*-values of these lexical property t-tests can be used as a reasonable proxy for similarity. Optimal solutions are those with the highest minimum *p*-value among the set of all pairwise t-tests for that solution; in other words, after t-testing every word list against every other list for each lexical property. Strictly speaking, *p*-values are not statistically defined or intended as a similarity measure; however, for practical purposes, they work well enough, and they have the useful property of having a clearly defined range (0–1) that is the same across different data types and scale factors. Using our GA, we were able to balance the words in all of our lists in such a way that a t-test between any two lists, and of any of the 5 relevant parameters, returned at *minimum* a test statistic of $p = .98$.

The GA-based balancing is implemented within our MATLAB toolbox LIBRA, which can be obtained via the MATLAB Central File Exchange (https://www.mathworks.com/matlabcentral/fileexchange/79628-lexical-item-balancing-resampling-algorithm-libra). Our starting word bank was obtained via the English Lexicon Project (ELP; [5]), an open repository of over 64,000 words that provides lexical property (e.g., number of syllables, number of phonemes) information, subjective norming data (e.g., valence), and behavioral measures such as speeded naming and LDT response times. An optional step in the LIBRA toolbox can be used to filter word banks obtained from the ELP, selecting subsets of the overall word bank that fit user-defined parameters. For example, one may wish to find only root words and filter out any homophones or compound words that also include the root. This filtering step offers versatility in constraint options beyond that of the ELP interface and automates many of the processes that researchers use to manually curate word lists. In order to make the toolbox functions more widely accessible, we have added a graphical user interface (GUI).

We will first examine the mechanics of how the GA approaches the balancing task, followed by detailed instructions for using the GUI. The toolbox has been tested to work with MATLAB 2015b through 2020a, but may work in other versions as well. The LIBRA toolbox does not require any additional add-on MATLAB toolboxes, only the base MATLAB installation.

Genetic algorithm implementation

The main GA script is fed an omnibus word bank along with the lexical parameters to be balanced (e.g., word length) and their associated values. A vector of list lengths specifies the number and size of each of the sub-lists to be generated. A target *p*-threshold is specified as the stopping point; this is the minimum *p*-value that will be allowed among all tests when evaluating whether a potential solution is good enough to be returned to the user. The GA will continue to run until either the *p*-threshold is attained, or a user-defined maximum number of iterations is reached. The GA first draws a potential solution, the first parent, at random from the omnibus word bank. A second solution, the

child, is created by cloning the parent and swapping one of the words from one of its sub-lists at random for one of the unused words in the omnibus bank. The fitness of both possible solutions is then independently evaluated. Each of the possible pairwise comparisons between sub-lists, for each parameter to be balanced, is then t-tested with an independent-samples t-test, resulting in a vector of *p*-values for each solution that describes its fitness. If the lowest *p*-value of the child solution is higher than the lowest *p*-value in the parent solution, the child is deemed to have better fitness and becomes the new parent; its genes (word lists) will become the starting point for the next generation (iteration of the GA). If the child solution fails to outperform the parent, it is discarded, and the parent will be re-used on the next generation. This process repeats until the lowest *p*-value of a solution has met the user-specified target *p*-threshold. Thus, over time the retained solutions are only allowed to stay the same or improve, but never become worse. Every 1000 iterations (about every 20 s in our test case, although timing can vary widely depending on the complexity of the problem and the hardware used), information about the fitness of the current solutions is printed to the command line to update the user.

Note that the GA applied here is relatively simple, and thus it has certain limitations that a more elaborate GA might not. For example, the GA could become trapped in a local minimum wherein swapping *any* single word will not result in an improved fitness score. However, given that most users will have a fairly large word bank relative to the desired number of stimulus words, this situation would be relatively rare. As such we have elected not to implement a more complex GA, which keeps the code simpler and limits the potential for bugs. Instead, if the GA performs a user-defined number of successive iterations with no fitness improvement, it is assumed that the algorithm has become stuck in a local minimum, and it will be forced to restart from the beginning. If a user-defined number of these restarts occurs, the script will terminate entirely, under the assumption that this means a valid solution is impossible (or impractically improbable) for the current set of parameters. If this occurs, users can modify the iterations or restarts variables, lower the *p*-threshold, modify their word bank, and/or discard one of their lexical parameters, and then try again. They also have the option to save the best solution found prior to termination, at which point they may deem the solution acceptable and choose to use it after all or use that information to guide them in adjusting their parameters. Note that failed attempts to reach the specified threshold are especially likely when the number of words used in a solution is high relative to the size of the unused word pool available for swapping.

When the algorithm achieves its threshold target, it will automatically save out a number of files. The first of these will be a MATLAB data file (.mat extension) containing each of the sub-lists generated, the *p*-value fitness metrics for each, and the original word bank along with an added column denoting which sub-list (if any) each word was assigned to; the final values for all of the other variables in the script are also retained for record-keeping purposes. A series of comma-separated value (CSV) files will also be generated, one for each wordlist, and the filenames will be appended with the list length and a unique number to allow differentiation (e.g., “my_wordlist_1_200.csv”, “my_wordlist_2_200.csv”).

The following instructions for use and validation are based on the application in Lintz and Johnson [14], a fairly typical usage example. To obtain an omnibus work bank, we requested a list of words from the ELP with the following constraints: between 3 and 10 letters, 1 to 2 syllables, 2 to 10 phonemes, a log HAL (Hyperspace Analogue to Language; [15]) frequency range of 2 to 11, and an average time to read aloud of between 500 and 1000 ms.

Graphical user interface (GUI) and usage

The toolbox comprises two scripts; one is the main GA that performs the list balancing, and the other provides functionality for filtering/trimming long omnibus word lists based on several simple criteria, prior to feeding the omnibus word list into the main GA script. The filtering script is optional; however, many users of ELP-based lists will find it useful for pruning unwanted words. The main GA script will accept a word bank from any source, as long as it is formatted properly. We will describe the usage of the main balancing script first (although if the optional filtering script is used, it would be run prior to the balancing script).

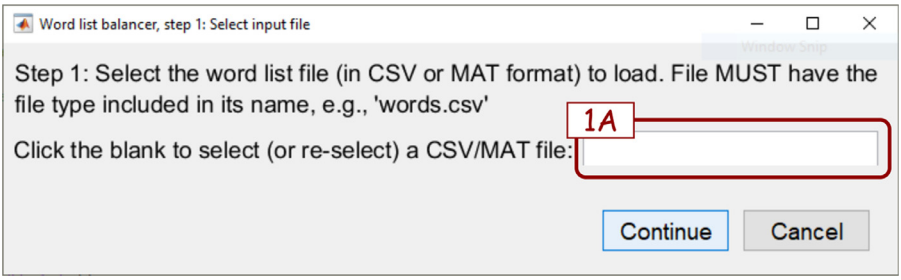

Fig. 1. Word list balancer, Step 1: Select input file.

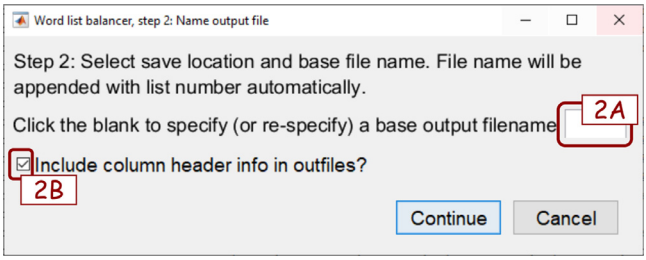

Fig. 2. Word list balancer, Step 2: Name output file.

Balancing script

Upon running the main GA script (libra_balance.m), a series of interactive MATLAB figure windows will appear, which we will describe individually.

Word list balancer, Step 1: Select input file (Fig. 1). First, the user selects a file to load for balancing (Fig. 1A); this is the omnibus word bank. Within the file selection window, users can choose between two file types, either CSV (default) or MAT (available after enabling that file type within the file selection window). If a MAT file is chosen, it is assumed to be the output of the optional filtering script (see next section), though any MAT file may be passed in as long as the wordlists contained therein are formatted the same and stored in the variable "list_to_balance". Word banks in CSV files are expected to conform to the following format: header information in the first row of each column, words in the first column, and each remaining column containing a lexical parameter to balance, which must consist of only numeric data. Although including too many parameters may limit performance, the script typically runs fairly quickly, and thus it is recommended to start with all of the desired parameters for balancing and a high *p*-threshold target, and only to reduce the threshold or cut back on the number of parameters if balancing fails.

Word list balancer, Step 2: Name output file (Fig. 2). Next, the user will specify a save location and filename stem for the output wordlists (Fig. 2A). Clicking in the dialog box will launch a standard file management window. Filename stems supplied by the user will be appended with a list number and list length for each wordlist that is output. Next, the user indicates via checkbox (Fig. 2B) whether to include column header descriptors in the output files (default is 'yes').

Word list balancer, Step 3: All other options (Fig. 3). Finally, the user will specify balancing parameters. First, the user enters the number of lists to generate and the length of each list. Lists need not be all the same size, though there are implications for finding an optimal solution if the lists differ greatly in length. Within the 'Wordlist sizes to create' text entry field (Fig. 3A), a list length is entered in a new row for each list desired (i.e., the default entry, [200; 200; 50], would create 2 lists of 200 words and one list of 50). In the next text field, the user enters the minimum *p*-value

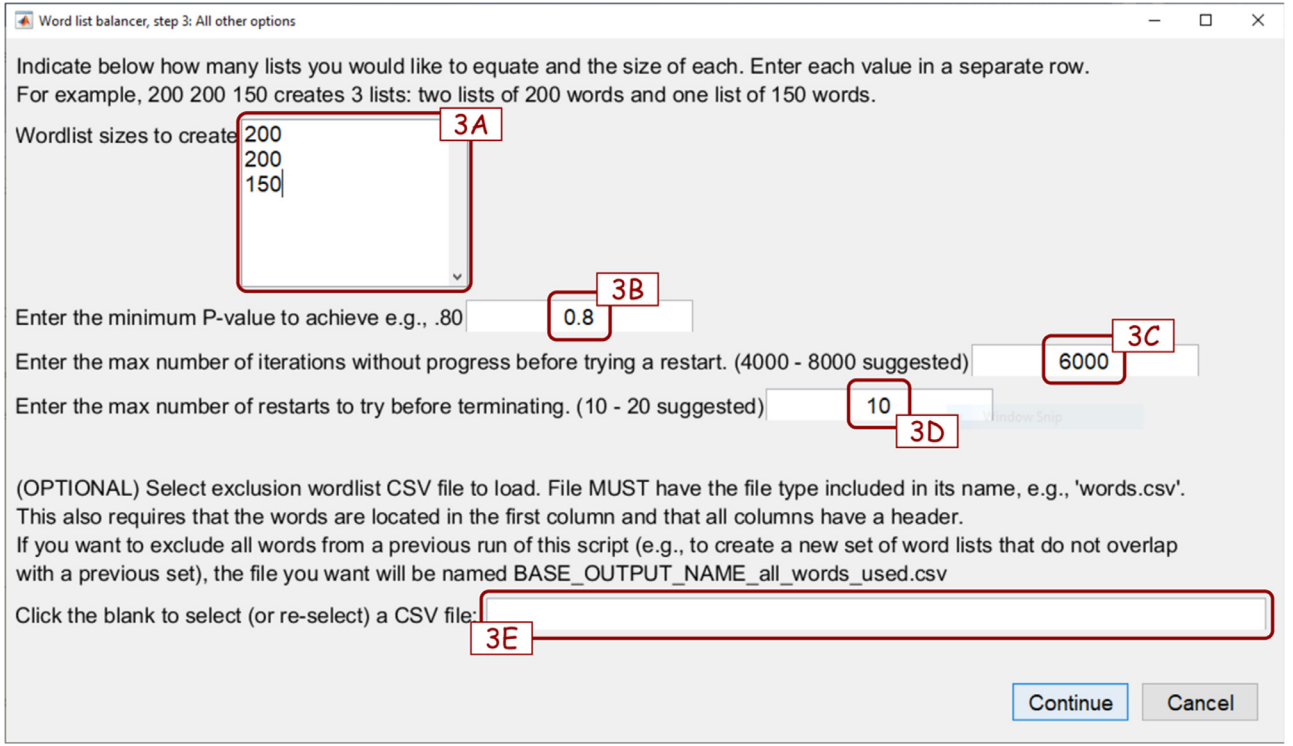

Fig. 3. Word list balancer, Step 3: All other options.

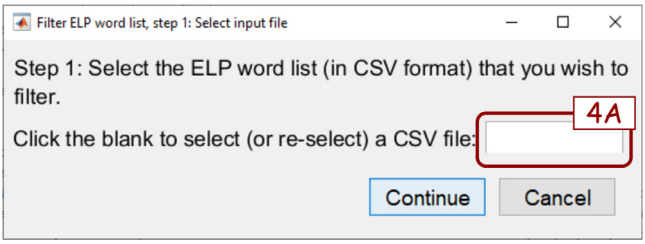

Fig. 4. Filter ELP word list, Step 1: Select input file.

desired (Fig. 3B); the default is 0.80, but for many applications, higher thresholds should be easily attainable, and it is recommended to first try a more optimistic value of 0.95 or higher.

The theoretically optimal solution will have an unknown *p*-value ceiling that depends on the number of lists, their lengths, the size of the available word pool, and the variance in the parameters being balanced. It is difficult to calculate or guess this ceiling *a priori* (tantamount to finding the optimal solution itself; hence the need for a randomization-based optimization technique in the first place), so a small amount of trial and error may be required to find a reasonable target threshold for each new study. In our validation tests (see 'Validation' section below), $p > .98$ was achievable for every permutation of 2–5 lists at 100, 200, 300, and 400 words per list.

Next, the user will enter the maximum number of GA iterations without progress before initiating a restart (Fig. 3C). A range of 4000–8000 is suggested based on our testing, and the default value for this setting is 6000. As *p*-values increase with successive iterations, it is increasingly likely that individual list swaps will not improve fitness and thus, if the number of iterations without forward progress reaches this value, it is assumed that a local minimum has been encountered and the GA will restart. Users will then enter the maximum number of such restarts to attempt (Fig. 3D) before the GA terminates.

Finally, an optional exclusion wordlist may be loaded (Fig. 3E). If an exclusion list is supplied, the stimulus sets generated and balanced will not include any of those words. This may be desired if, for example, the user wishes to conduct follow-up experiments or add conditions at a later point that are derived from the same initial word bank but should not re-use any of the words previously used in a solution. The exclusion list can either be the CSV file that was output from a previous balancing operation, or any other CSV file formatted similarly.

Output files from the balancing script include: Both CSV and MAT versions of each balanced list requested, a CSV of all words included in all balanced lists, a CSV of the initial wordlist with list assignments appended, a text file of the final fitness values, and a MAT file containing the end state of all of the variables in the balancing script.

Filtering script

This optional script (libra_filter.m) provides simple filtering of word banks obtained from the ELP. The restriction to ELP-sourced CSV files is due to reliance on the specific header names and the syntax used in certain ELP data columns, though in theory any CSV file can be filtered as long as the corresponding data columns adhere to the same syntax. If this script is used, it should be applied prior to the balancing script. A number of pruning options are available (e.g., removing proper nouns, removing homophones), which are typically faster and/or more reliable than doing the same by hand. It should be noted that while these options are helpful, some are based on simple heuristics that may not recognize certain edge cases, so manual review of the final filtered list is highly recommended. Running the filtering script will prompt a series of interactive windows, as in the balancing script.

Filter ELP word list, Step 1: Select input file (Fig. 4). First, the user will select the CSV file of words to filter. Clicking in the dialog box (Fig. 4A) will launch a standard file selection window.

Filter ELP word list, Step 2: Name output file (Fig. 5). Next, the user specifies the name and location of the output file with the dialog box (Fig. 5A), similarly to the balancing script.

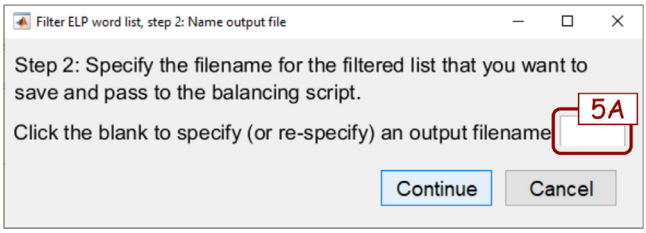

Fig. 5. Filter ELP word list, Step 2: Name output file.

Filter ELP word list, Step 3: Acceptable parts of speech (Fig. 6). The first filtering option is to select the part(s) of speech (POS) for words retained in the final list. Users may choose from three filtering methods (Fig. 6A). The default option, "selections in boxes," allows users to select for single POS or multiple combinations of POS. Alternatively, selecting "all possible combos" will apply all possible permutations of POS combinations. Users may also choose to not filter by POS. In most cases, this last option will behave the same as if filtering is performed with all possible combinations of POS; however, this option may be useful in the case of wordlists without POS data. Steps involving filtering for POS require that the POS column is present in the ELP file. If any individual words lack POS data, they will be removed from the list automatically.

Filter ELP word list, Step 4: Types of words to remove/flag (Fig. 7). The next set of options is to remove words based on specific criteria (Fig. 7A–7D), flag them for manual review (Fig. 7E–7H) rather than remove them automatically, or both remove them and flag those words as removed for future reference. As with the POS options, if these steps are selected, any words missing the relevant data will automatically be removed.

One such option is to remove variants, defined as words that share a stem with other words in the list (e.g., *abandon, abandoned*, and *abandoning*). It is often undesirable to have such closely related words in the final stimulus list. If variant removal is selected (Fig. 7A), the first occurrence of a variant will be retained while subsequent occurrences are removed. If users wish to manually select which of the variants to remove, they can choose 'no' for removal and instead flag them (Fig. 7E) for later manual review. The identification of variants requires the ELP's 'MorphSp' (word spelling based on morphemes) column to be present in the CSV file. Note that variant identification uses a fairly simple heuristic that identifies matches among morphemes, and thus it may miss certain edge cases. This is deliberate, as tightening the algorithm would also considerably increase the number of flagged words based on small segments of those words that share the same spelling, but appear within words unrelated in meaning (e.g., **cat**alog and **cat**aract).

Similar to the rationale for removing variants, homophones may also be problematic. Homophone removal (Fig. 7B) and flagging (Fig. 7F) are both optional and function similarly to variant removal. Identifying homophones requires the 'Pron' (pronunciation) column to be in the CSV file. The script is designed to retain the first occurrence of a word sharing the same pronunciation as another in the list and remove/flag all subsequent occurrences. The pronunciation data from the ELP is based on a standardized, computer-readable, phonetic alphabet for American English, and thus the homophone identification behavior tends to be quite reliable.

Proper nouns such as the names of people and cities are fairly common in ELP-sourced lists and can be removed (Fig. 7C) and/or flagged (Fig. 7G) as in previous steps. Since identifying proper nouns (operationalized as any word that is capitalized) is straightforward, performance in this step is robust.

Lastly, users may choose to remove (Fig. 7D) and/or flag (Fig. 7H) compound words. There may be no intrinsic reason to exclude a word simply because it is compound; however, some may find it advisable, as it prevents two or more words appearing in the same stimulus set that share a word stem (e.g., cat, walk, and catwalk). Like variant removal, identifying compound words requires the 'MorphSp' column to be in the CSV file. In that column, word stems appear inside of curly braces, and thus, this step identifies words with multiple sets of braces. If compound words are acceptable and the user is only concerned about them appearing along with their constituent stems, selecting

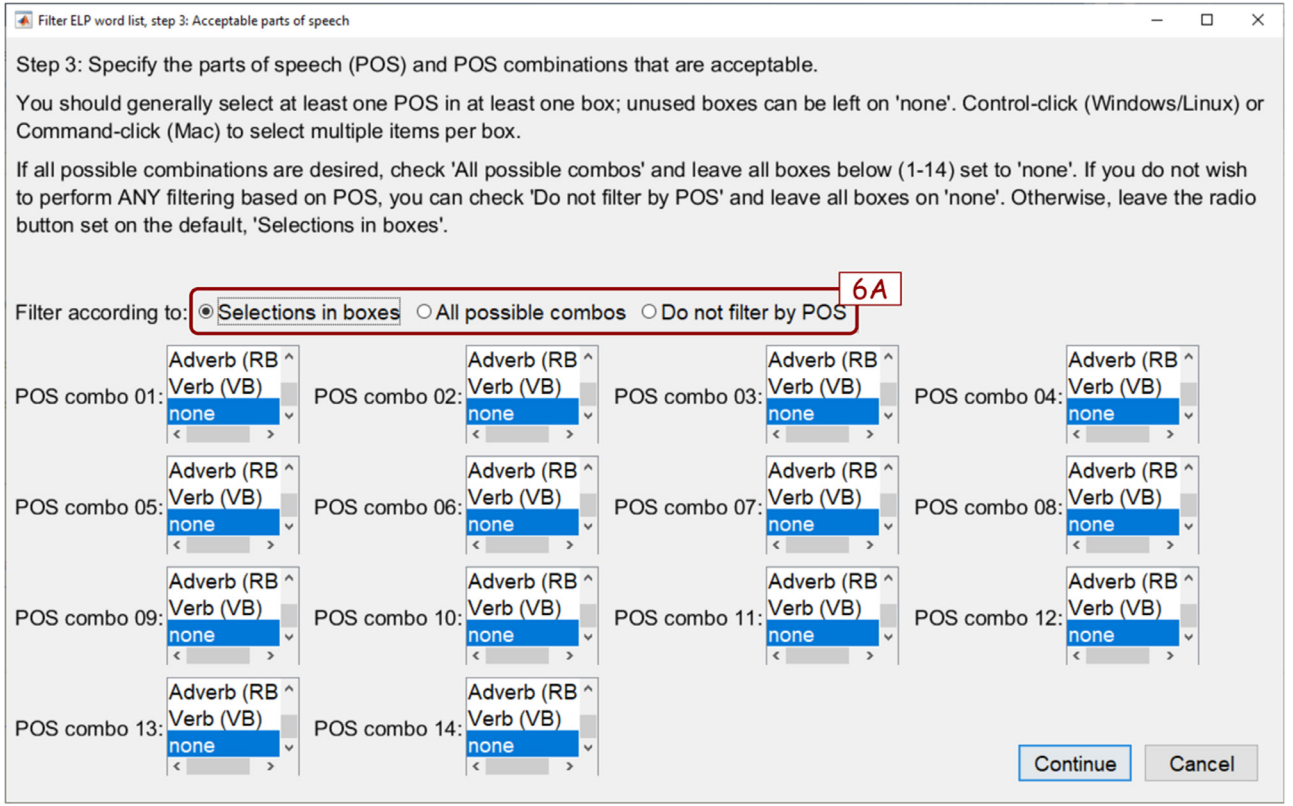

Fig. 6. Filter ELP word list, Step 3: Acceptable parts of speech.

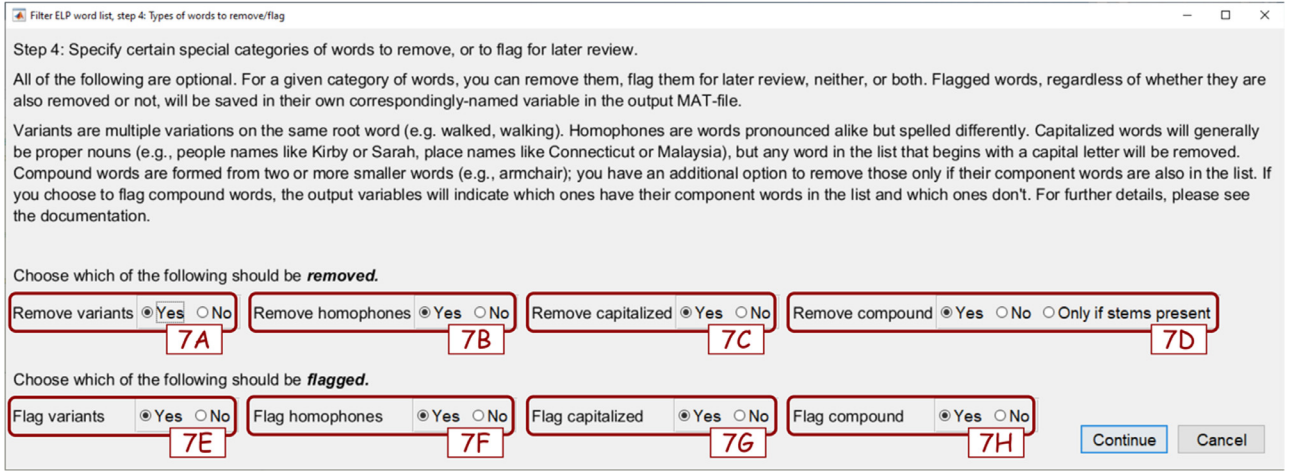

Fig. 7. Filter ELP word list, Step 4: Types of words to remove/flag.

the 'only if stems present' option then evaluates the contents of each set of braces, comparing it to all others in the list. Note that many of the word stems may have already been excluded in the previous steps, and therefore choosing this option will only remove a compound word if one of its stems is actually present in the current pruned version of the list.

Filter ELP word list, Step 5: Choose columns to include in output (Fig. 8). Finally, the user selects the data columns for balancing. As the output of the filtering script is intended to be passed to the GA script, which then balances all of the columns present, this step retains all checked columns while stripping the unchecked columns from the list that is saved out. Across all of the filtering steps, the imported ELP CSV file is never modified; 'removal' only applies to the output that is created. Nearly all of the ELP columns in the input file with numeric data are potentially available for selection, although most experiments will only require a handful of parameters to be considered. If the input file includes any data columns that are not ELP-standard, users may select 'Also include user-defined properties' to balance by the numerical data in those columns. Of course, one should have a reasonable methodological justification for balancing by any parameter; while balancing by certain parameters might be theoretically possible, the result may not necessarily be meaningful in practice. Users should consult Balota et al. [5] for descriptions of each parameter to help them decide which are most relevant for their studies. It is also good to keep in mind that, as noted previously, including more columns will likely increase the time required to find an adequately balanced solution; in extreme cases, including too many columns could preclude finding a solution at all.

Another factor to consider when including/excluding columns is the variance present for some parameters. Given that t-tests are employed as a fitness function, high variance may produce a low ceiling for the fitness of the optimal solution. For this reason, among others, though one could balance by the HAL frequency norms (denoted as Freq_HAL in the ELP), it makes more sense to use the log-transformed version (Log_Freq_HAL) instead; the same consideration should be given to similar cases.

Once all of the desired options have been selected, clicking on 'Continue' will apply those filtering operations and save the results to a MAT file. Within that output file, all of the variables from the filtering script are retained for future reference. If users select any of the flagging options, those words that are flagged will be marked as 'true' in the corresponding variable names beginning with the word 'flag' (e.g., flag_capitalized). Once more, manual inspection of the final filtered word bank is highly recommended. As we have previously discussed, some filtering steps rely on simple heuristics to identify target words for filtering, and thus it may be possible for some edge cases to be missed. This is necessary in part due to the format of the data supplied in ELP spreadsheets, and in part to simplify the code. In addition, other undesirable outcomes are possible that even a complicated algorithm might miss despite being immediately apparent to a human reviewer (e.g., 'catsup' and 'ketchup' appearing together).

Validation

To put LIBRA into practice, we ran two validation tests across a range of stimulus list lengths and number of lists that might be commonly used in cognitive psychology experiments. First, we requested a word bank from the ELP and ran it through the LIBRA filtering script. In Validation 1, we set the LIBRA balancing script to generate multiple combinations of stimulus lists that were balanced on five lexical properties, finding that five lists of 400 words each could be created in under 20 min. In Validation 2, we created lists by randomly selecting words from the word bank, and demonstrated that the vast majority of these randomized lists differed significantly on at least one lexical property.

Word bank filtering

A word bank was requested from the ELP (restricted set; ELP experiment norms rather than the HAL-derived norms) list with all options checked for inclusion. Additionally, constraints or ranges were specified for certain lexical properties as follows: Word length (3–10 letters), log HAL frequency (2–11), number of phonemes (2–10), number of syllables (1–2), and average time to read aloud (500–1000ms). These criteria were similar to the stimulus set used by Lintz and Johnson [14] and chosen so that validation would be representative of an actual experiment. These selections returned an initial word bank of 20,509 words, which was subsequently filtered using libra_filter.m with the

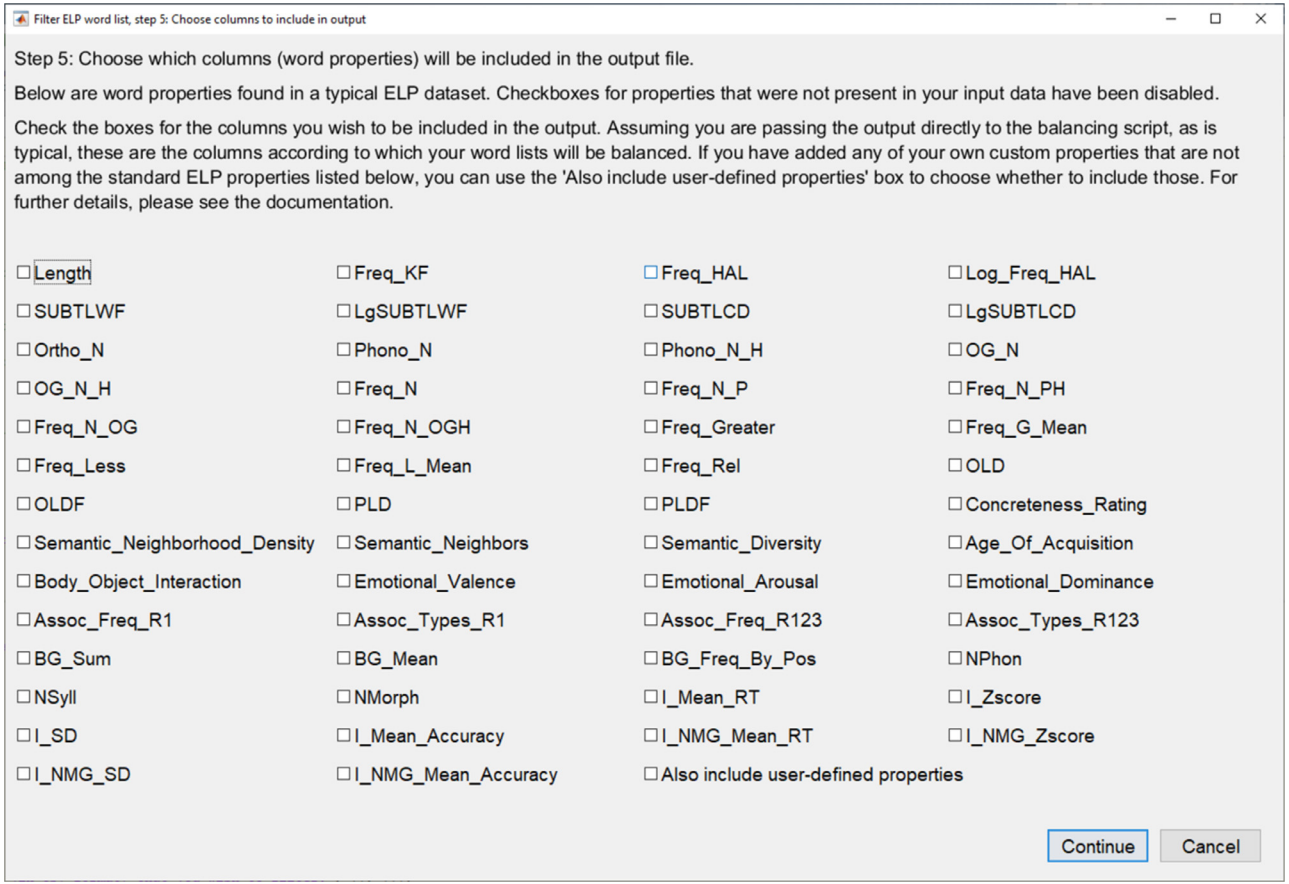

Fig. 8. Filter ELP word lists, Step 5: Choose columns to include in output.

following options: Nouns only ('NN', single POS); proper nouns, variants, and compound words (if stems present) were removed. Words with multiple POSes were also removed. The resulting filtered list contained 3,987 words and retained for balancing purposes the columns corresponding to number of letters, log HAL frequency, number of phonemes, number of syllables, and time to read aloud.

Validation 1: Generate LIBRA-balanced word lists and measure runtime

The algorithm from the libra_balance.m script was then used to generate multiple combinations of stimulus lists that were balanced across the five lexical properties in the final filtered word bank described above. Four stimulus list lengths were used (100, 200, 300, and 400) and four stimulus list numbers were used (2, 3, 4, and 5), resulting in 16 different possible combinations (two, three, four, and five lists of 100 words each; two, three, four, and five lists of 200 words each, and so on) representing a range of what might be reasonably expected for a typical psychology experiment using word stimuli. The target minimum p-value was set to $p > .98$. Maximum allowable iterations and restarts were set at 6,000 and 10, respectively. Each of the 16 possible combinations was repeated 50 times to obtain a reliable measure of mean running time for an average desktop computer.

Validation 2: Baseline comparison technique – purely randomized stimulus selection

To provide a comparison to the LIBRA-balanced lists, a Monte Carlo process was used to perform random draws from the same filtered word bank of 3,987 words used in the previous Validation 1. This simulated the outcome of assigning words to condition lists for each of the same 16 combinations of list length and number of lists, but instead of balancing lexical properties, purely random assignment (another common experimental technique) was used. These random draws were performed 10,000 times per combination in order to establish a reliable average. For each iteration, the degree of similarity between sub-lists was tested using the same criteria as in the balancing script: Every sub-list was t-tested against every other sub-list, for all five lexical properties. This resulted in as few as five t-tests (for sets containing two sub-lists) and as many as fifty (for sets containing five sub-lists) per iteration.

Validation 1 results: Generate LIBRA-balanced word lists and measure runtime

Good balancing among sub-lists was achieved in every case we examined. All combinations of list length and number of lists returned a *minimum* p-value of .98 among all tests performed when up to four wordlists were generated (with one exception among four-list tests, representing 2% of those tests). When five lists were generated, 26.5% of balancing attempts contained at least one t-test that did not return a minimum p-value of .98 before exceeding the specified allowable number of restarts, though all t-tests returned a minimum p-value of at least .85.

Thus, balancing in even the most challenging (five-list) scenario was highly successful. First of all, even for iterations that "failed" to reach criterion, $p > .85$ for all tests still represents an eminently usable degree of similarity among sub-lists and, in our experience, far exceeds the balancing any human could achieve by hand. Second, the fact that a high percentage of attempts *did* reach the $p > .98$ target indicates that it is possible to achieve such balancing in a reasonable time frame, just not on every single iteration. So, in a real-life situation, a researcher could easily obtain near-perfectly balanced lists more often, merely by relaxing the timeout parameters or running more iterations. Fig. 9 illustrates the average time required to reach the target fitness p-value for the 16 combinations tested, which span a range of use cases that should encompass most cognitive experiments' needs.

Even for the most demanding scenarios of five wordlists, fitness targets were achieved in an average of 18.3 min; smaller set sizes were balanced considerably faster (within a matter of seconds for sets of 2 or 3 wordlists; under 4 min for sets of 4 wordlists).

Validation 2 results: Baseline comparison technique – purely randomized stimulus selection

In contrast to the LIBRA-balanced lists, in 46.7% of the generated list sets using purely random selection, there was at least one pair of wordlists that differed significantly (at the conventional threshold of $p < .05$) on one of the five lexical properties. As might be expected, we see more of these "significant" differences in sets containing more lists, and to a lesser extent, sets with longer lists.

Table 1 gives the percentage of Monte Carlo simulations for each combination in which the minimum p-value obtained during testing was less than .05. It is important to note that the p-values here cannot be interpreted exactly in the usual way, as the probability of observing a given result under the null hypothesis. In these simulations, we *know* the null hypothesis is true, because we

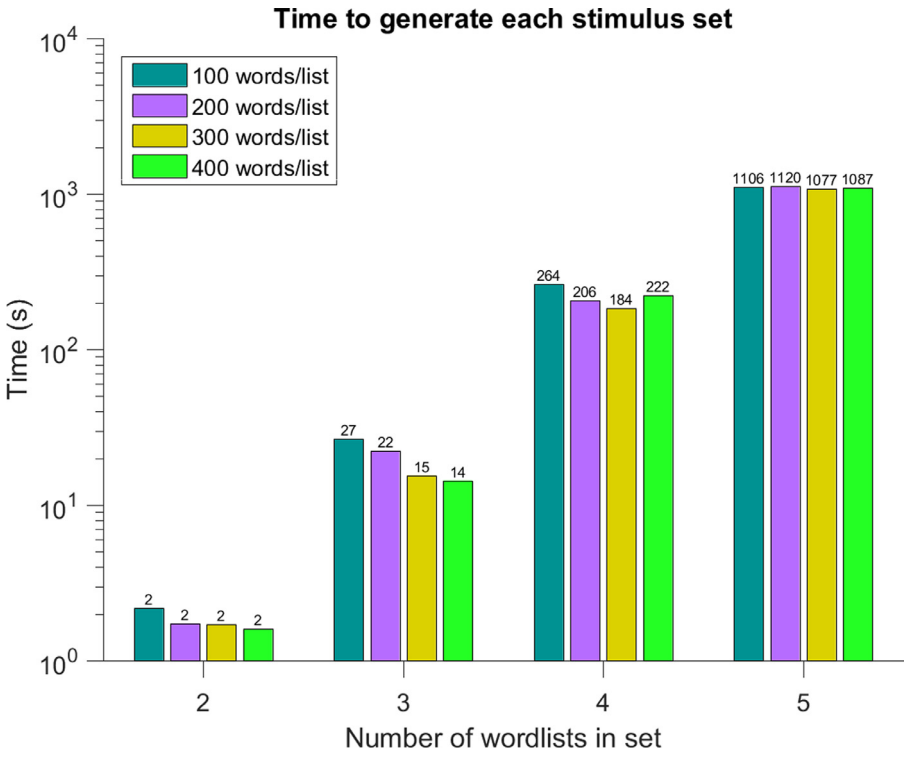

Fig. 9. Time to generate each stimulus set for validation 1 using the libra_balance algorithm. Sixteen stimulus sets were generated, balanced across five lexical properties. Target minimum p-value was set to .98, which was achieved in the vast majority of cases; in all cases, a minimum p-value of at least .85 was achieved.

Table 1
Percentage of randomized lists with at least one "significant" inter-list difference ($p < .05$).

	Number of wordlists				
List Length	2	3	4	5	Mean
100	18.4	39.9	56.8	71.4	46.6
200	19.1	38.5	57.6	71.7	46.7
300	17.9	38.7	57.5	72.7	46.7
400	18.1	39.7	57.1	72.4	46.8
Mean	18.4	39.2	57.2	72.0	46.7

intentionally drew the lists from the same underlying population. Here, we are using *p*-values as convenient proxies for how well-equated (or not) two lists are, so rather than using the conventional language of statistical "significance," it might be more apt to say that list pairs with $p < .05$ are "substantially" unbalanced. These results suggest that randomly assigning words to conditions will result in substantially unbalanced stimulus lists between 17.9% and 72.7% of the time. As is evident in Table 1, list length differences tend to produce similar results, with the number of word lists being generated driving most of the resulting variance in performance. Thus, we have collapsed across list length to present the results shown in Fig. 10, which shows histograms of the smallest *p*-values obtained for each iteration of validation 2 at each set size.

It is only fair to note that even if two lists differ substantially on one lexical property (e.g., length), it is certainly possible to have differences on other properties (e.g., frequency) that contribute

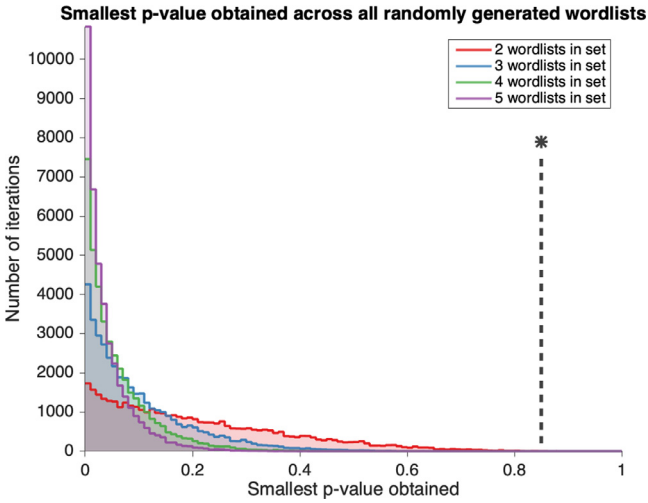

Fig. 10. Smallest p-value obtained across all randomly generated wordlists in validation 2. The dotted line and asterisk represent the worst performance achieved across all iterations of the genetic algorithm. Virtually none of the randomized Monte Carlo simulations achieved a level of performance equal to the worst LIBRA performance.

bias in the opposite direction, such that the biases cancel each other out. Likewise, most well-designed experiments of this nature would not simply perform one such randomization; typically, word lists would be re-randomized (or at least re-assigned to other conditions in a counterbalanced fashion) across experimental subjects. Thus, with a large enough participant sample, the law of large numbers should ensure that there is no *systematic* bias in the stimuli. However, it is still vastly preferable to give *each* subject as well-balanced a set of lists as possible, for two reasons. First, failing to balance lists well within subjects will drastically inflate the variance between those subjects (with some having extreme bias in one direction, some having low bias, and still others having extreme bias in the other direction), and it will correspondingly lower statistical power. Second, sufficiently unbalanced lists could have secondary effects well beyond the low-level influences of the lexical properties on measures like response time. For instance, if a participant becomes consciously aware that one condition's words are consistently longer than another's, they might change their strategy, suspect deception, lose focus on the task, or behave in other unpredictable ways that could distort the results to a degree that violates the typical expectations underlying law-of-large-numbers logic.

Conclusion

Here we have introduced a method and toolbox for equating the lexical properties of word lists used in cognitive experiments. We have demonstrated and quantified how relying on purely randomized procedures to assign word bank items to lists often results in substantially unbalanced experimental conditions. In contrast, our method guarantees that the lists used for each experimental condition contain no meaningful differences, and thus that the lexical characteristics of the specific words used will add an absolute minimum of bias/noise to the experiment in which they are used.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] B.C. Armstrong, C.E. Watson, D.C. Plaut, SOS! An algorithm and software for the stochastic optimization of stimuli, Behav. Res. Methods 44 (3) (2012) 675–705, doi:10.3758/s13428-011-0182-9.

[2] A.D. Baddeley, N. Thomson, M. Buchanan, Word length and the structure of short-term memory, J. Verbal Learn. Verbal Behav. 14 (1975) 575–589.

[3] D.A. Balota, J.I. Chumbley, The locus of word-frequency effects in the pronunciation task: lexical access and/or production? J. Mem. Lang. 24 (1) (1985) 89–106.

[4] D.A. Balota, M.J. Cortese, S.D. Sergent-Marshall, D.H. Spieler, M.J. Yap, Visual word recognition of single-syllable words, J. Exp. Psychol. Gen. 133 (2) (2004) 283–316, doi:10.1037/0096-3445.133.2.283.

[5] D.A. Balota, M.J. Yap, K.A. Hutchison, M.J. Cortese, B. Kessler, B. Loftis, J.H. Neely, D.L. Nelson, G.B. Simpson, R. Treiman, The English Lexicon Project, Behav. Res. Methods 39 (3) (2007) 445–459, doi:10.3758/BF03193014.

[6] C. Burgess, K. Livesay, The effect of corpus size in predicting reaction time in a basic word recognition task: moving on from Kučera and Francis, Behav. Res. Methods Instrum. Comput. 30 (2) (1998) 272–277.

[7] F.M.M. Citron, B.S. Weekes, E.C. Ferstl, Arousal and emotional valence interact in written word recognition, Lang. Cogn. Neurosci. 29 (10) (2014) 1257–1267, doi:10.1080/23273798.2014.897734.

[8] M. Coltheart, The MRC psycholinguistic database, Q. J. Exp. Psychol. 33 (4) (1981) 497–505, doi:10.1080/14640748108400805.

[9] M. Guasch, J. Haro, R. Boada, Clustering words to match conditions: An algorithm for stimuli selection in factorial designs, Psicol. Int. J. Methodol. Exp. Psychol. 38 (1) (2017) 111–131.

[10] J.A. Higgins, M.R. Johnson, M.K. Johnson, Age-related delay in reduced accessibility of refreshed items, Psychol. Aging 35 (5) (2020) 710–719, doi:10.1037/pag0000458.

[11] M.R. Johnson, J.A. Higgins, K.A. Norman, P.B. Sederberg, T.A. Smith, M.K. Johnson, Foraging for thought: an inhibition of return-like effect resulting from directing attention within working memory, Psychol. Sci. (24) (2013) 1104–1112, doi:10.1177/0956797612466414.

[12] E. Keuleers, D.A. Balota, Megastudies, crowdsourcing, and large datasets in psycholinguistics: an overview of recent developments, Q. J. Exp. Psychol. 68 (8) (2015) 1457–1468, doi:10.1080/17470218.2015.1051065.

[13] H. Kučera, W.N. Francis, Computational Analysis of Present Day American English, Brown University Press, Rhode Island, 1967.

[14] E.N. Lintz, M.R. Johnson, Refreshing and removing items in working memory: different approaches to equivalent processes? Cognition 211 (2021) 104655, doi:10.1016/j.cognition.2021.104655.

[15] K. Lund, C. Burgess, Producing high-dimensional semantic spaces from lexical co-occurrence, Behav. Res. Methods Instrum. Comput. 28 (1996) 203–208, doi:10.3758/BF03204766.

[16] E.L. Thorndike, I. Lorge, The Teacher's Word Book of 30,000 Words, Bureau of Publications, Teachers Co, 1944.

[17] M. van Casteren, M.H. Davis, Match: a program to assist in matching the conditions of factorial experiments, Behav. Res. Methods 39 (4) (2007) 973–978, doi:10.3758/BF03192992.