Behav Res
DOI 10.3758/s13428-011-0182-9

SOS! An algorithm and software for the stochastic optimization of stimuli

Blair C. Armstrong · Christine E. Watson · David C. Plaut

Abstract The characteristics of the stimuli used in an experiment critically determine the theoretical questions the experiment can address. Yet there is relatively little methodological support for selecting optimal sets of items, and most researchers still carry out this process by hand. In this research, we present SOS, an algorithm and software package for the stochastic optimization of stimuli. SOS takes its inspiration from a simple manual stimulus selection heuristic that has been formalized and refined as a stochastic relaxation search. The algorithm rapidly and reliably selects a subset of possible stimuli that optimally satisfy the constraints imposed by an experimenter. This allows the experimenter to focus on selecting an optimization problem that suits his or her theoretical question and to avoid the tedious task of manually selecting stimuli. We detail how this optimization algorithm, combined with a vocabulary of constraints that define optimal sets, allows for the quick and rigorous assessment and maximization of the internal and external validity of experimental items. In doing so, the algorithm facilitates research using factorial, multiple/mixed-effects regression, and other experimental designs. We demonstrate the use of SOS with a case study and discuss other research situations that could benefit from this tool. Support for the generality of the algorithm is demonstrated through Monte Carlo simulations on a range of optimization problems faced by psychologists. The software implementation of SOS and a user manual are provided free of charge for academic purposes as precompiled binaries and MATLAB source files at http://sos.cnbc.cmu.edu.

Keywords Stimulus selection · Internal validity · External validity · Factorial designs · Multiple/mixed-effects regression designs · Constraint satisfaction · Stochastic optimization · Monte Carlo simulation

B. C. Armstrong (✉) · D. C. Plaut
Department of Psychology and Center for the Neural Basis of Cognition, Carnegie Mellon University,
5000 Forbes Avenue,
Pittsburgh, PA 15213, USA
e-mail: blairarm@andrew.cmu.edu

C. E. Watson
Department of Neurology and Center for Cognitive Neuroscience, University of Pennsylvania,
3400 Spruce Street,
Philadelphia, PA 19104, USA

Published online: 21 February 2012

All empirical researchers recognize that a fundamental challenge in the preparation of a new experiment is the selection of stimuli that are optimally suited to address the theoretical question of interest. The definition of what constitutes an optimal set can be unpacked into two main components: (1) The manipulation of the variable(s) of interest should be as large as possible in the absence of confounds with other variables (i.e., internal validity), and (2) the stimuli should be representative of their underlying population(s), thus permitting inferences to items not present in the experiment (i.e., external validity). Researchers also know that the consequences of failing to select an adequate set of stimuli can be dire. In the worst-case scenario, suboptimal stimuli can generate putative theoretical "advances" and protracted periods of belief in incorrect theoretical positions (Cutler, 1981; Gernsbacher, 1984). In a less extreme scenario, suboptimal stimuli can make it impractical to achieve the necessary statistical power to study the effects of interest (e.g., Armstrong, 2007). Even in the best-case scenario, the use of suboptimal stimuli wastes time and other resources in efforts to increase the statistical power of the experiment, such as by running additional participants.

Given these issues, a newly minted psychologist would undoubtedly expect a high level of rigor in the methods available for selecting stimuli, particularly given the progress that has been made in other aspects of experimental design and analysis. For example, in visual word recognition

research, massive efforts have been devoted to identifying variables that influence performance (Cortese & Khanna, 2007), to creating databases and software that list the values of these variables for large sets of items (Brysbaert & New, 2009; Coltheart, 1981; Davis, 2005; Kučera & Francis, 1967), to increasing the precision of timing hardware and software (Brainard, 1997; Schneider, Eschman, & Zuccolotto, 2002), and to improving the statistical frameworks used to analyze data (Baayen, Davidson, & Bates, 2008; Clark, 1973; Raaijmakers, Schrijnemakers, & Gremmen, 1999). The state of the art with respect to the selection of experimental items would therefore come as a shock: the standard method for selecting stimuli is to do so by hand with the assistance of a sorting function in a spreadsheet application.

The aim of the present work is to bring methodological rigor to the stimulus selection process, with the ultimate goal of facilitating the development of experiments that are better suited for the empirical evaluation of researchers' hypotheses. To this end, we introduce SOS—an algorithm and software for the stochastic optimization of stimuli used in experiments, based on a variant of a classic stochastic relaxation search (Kiefer & Wolfowitz, 1952). We start by characterizing the benefits and drawbacks of existing manual selection heuristics and computational search algorithms that can be used to identify optimal experimental stimuli. We then briefly describe how SOS augments a simple manual optimization heuristic to rapidly and reliably discover optimized stimuli. This is accomplished by tailoring a general stochastic optimization algorithm so that it is well suited to the types of selection constraints and challenges to successful optimization faced by psychologists during item selection and other similar problems. Following the description of the algorithm, we report the results of Monte Carlo simulations of SOS performance in solving optimization problems that are encountered in a variety of experimental designs. These examples serve to illustrate the robustness of the algorithm and its superiority to manual stimulus selection, the new types of experimental designs it facilitates, and the increased rigor it brings to evaluating and optimizing the internal and external validity of experimental items. These simulations also demonstrate that our software implementation of the algorithm can be used with relative ease to identify optimal stimuli. Note that although all of these example optimization problems involve matching word sets for use in psycholinguistic experiments—an area in which such optimizations may be particularly valuable—the algorithm is domain general and can be applied to any situation that involves selecting from a population of items characterized on a number of dimensions. Extended uses of SOS, such as for the selection of control participants for neurologically impaired patients in case series analyses, are briefly discussed in the final section. A user manual including additional details of the SOS procedure, example optimizations, and the full, open-source software implementation of the algorithm as either MATLAB source files or precompiled binaries for major operating systems is available free of charge at http://sos.cnbc.cmu.edu.

A manual heuristic for identifying optimal stimuli

The SOS algorithm has much in common with a simple manual heuristic for identifying optimal stimuli; indeed, the simplest version of the algorithm ("greedy" optimization of stimuli, discussed later) basically amounts to a formalization of this heuristic. It can be broken down into the following major steps:

1. Determine the conditions that will be employed to study the variable(s) of interest.[1]
2. Define constraints that establish how variables should differ or be equated across conditions or within a condition.
3. Identify a population of items and fill each condition with a sample of these items.
4. Search for an item in the sample and an item in the population that could be swapped to better satisfy the constraints.
5. Evaluate the degree to which the constraints have been satisfied (e.g., run *t*-tests to confirm that the variables that should differ between conditions do, in fact, differ). Repeat the search and evaluation steps until the constraints have been satisfied to a target threshold or until some other reason to stop the search has arisen, such as simply deciding to use the best set found to date after many unsuccessful attempts at swapping items.
6. Assess the degree to which the sample stimuli are representative of the underlying populations from which they originated. This step is often ignored but is necessary to draw statistical inferences to the broader population of items. Specifically, it is possible that the constraints have limited the items that are included in the sample to some contorted and unusually distributed subpopulation of the original population. Evaluating whether this is or is not the case is critical for determining the correct statistical analyses and inferences that can be drawn on the basis of the selected stimuli (Baayen et al., 2008; Clark, 1973; Hino & Lupker, 1996; Raaijmakers et al., 1999).

[1] For the purpose of illustration, we have tried to keep examples simple and transparent by using well-known experimental designs. However, we recognize that other superior but less familiar designs may be more suitable for testing some of these hypotheses (see, e.g., Baayen et al., 2008, for alternatives to standard multilevel designs in the study of continuous variables). How SOS can facilitate the selection of stimuli in some of these alternative designs is discussed in later sections.

Problems with the manual heuristic

By and large, the manual heuristic just described is not unreasonable for selecting experimental stimuli. Nevertheless, it possesses several undesirable characteristics:

- The procedure is not actually optimal relative to automated optimization methods (support for this claim is provided later in this article and in van Casteren & Davis, 2007). This becomes increasingly evident as the complexity of the optimization problem increases. As was foreseen by Cutler (1981), the complexity of stimulus selection has only increased as time has passed, so these differences will likely be exacerbated in the future.
- The procedure is exceedingly tedious and time consuming and is, consequently, responsible for the reuse of existing items. As a result, the generalizability of effects is hampered, since the gold standard for generalization is replication with new items (Stanovich, 1997). The recycling of a particular set of stimuli can also lead to a single point of failure for a large body of work: If a problem is identified later with that particular set, the results of all the studies using those stimuli may be called into question. For instance, the ambiguous words used by Rodd, Gaskell, and Marslen-Wilson (2002), have been reused in studies by Armstrong and Plaut (2008) and Beretta, Fiorentino, and Poeppel (2005). However, the results of these studies were called into question when later work by Armstrong and Plaut (2011) identified additional variables that were not controlled for in these items. Issues with the Snodgrass and Vanderwart (1980) pictures raised by Bunn, Tyler, and Moss (1998) also cast doubt on the outcomes of many studies employing these items.
- The procedure is rarely followed up with an examination of whether the selected stimuli are representative of the populations from which they were sampled. The lack of this step is conceivably due to a combination of factors, including the lack of a standard means of evaluating representativeness and the lack of pressure for these analyses to be included when results obtained using a particular set of items are reported. This oversight is troubling because the pressure to satisfy the constraints may distort how well the population is represented in the sample. Thus, the results of an experiment run with a particular set of stimuli may not generalize to the population from which the stimuli were sampled; the group of items that can satisfy the constraints may consist of only a small and possibly atypical portion of the population. Although this problem may exist for many different types of constraints a researcher could impose on the optimization process, factorial designs that cross intercorrelated variables may be particularly at risk of being nonrepresentative (see Baayen et al., 2008, for a discussion). The lack of an evaluation of representativeness is also somewhat paradoxical, given the long-standing debates concerning the use of statistics to generalize the results of experiments both across items and across participants (Baayen et al., 2008; Clark, 1973; Hino & Lupker, 1996; Raaijmakers et al., 1999). These techniques are of little value if the population to which the results are being generalized is unknown.
- The procedure is prone to human error.

Given these problems, it is clear that the simple manual heuristic, although not without its merits, leaves much to be desired.

Alternatively, one avenue that intuitively might appear to address many of the problems with the manual heuristic outlined above would be to use a simple search algorithm from computer science to find optimal stimuli. However, these algorithms also tend to be unsuitable for selecting stimuli because exhaustive searches of all combinations of items are impractical and random searches of a subset of the possible combinations of items do not reliably identify optimal sets.

Stochastic optimization of stimuli: A brief overview

Considered as a whole, there are some advantages to both the human heuristic, which often yields stimuli that are sufficiently optimized for researchers to consider them suitable for use, and simple computational search algorithms, which allow very large numbers of combinations of stimuli to be examined but do not focus their efforts on exploring the combinations that are most likely to be optimal. A successful amalgamation of these two approaches might therefore leverage the unique advantages of both without the faults of either.

SOS is an attempt at such a synthesis. In its most basic form, the algorithm amounts to a translation of the steps from the manual stimulus selection heuristic into a "greedy" optimization search. In simple cases, this involves the following basic steps, although a range of more advanced options may also be applied. A detailed description of these steps, including their formal underpinnings, is presented in the user manual and is recommended reading for researchers who wish to take full advantage of the algorithm.

1. Define the samples (experimental conditions).
2. Specify constraints that establish the desired relationships between the different variables within and/or between the samples, and associate each constraint with a *cost* function that operationalizes violations of the constraint. A broad vocabulary of constraints has been

implemented in SOS that may be used to select items for a wide variety of experimental designs. These constraints are briefly described here, and the full details and formal underpinnings of the constraints are presented in Appendix 1.

Currently, the vocabulary of constraints can be divided into two main types: hard constraints and soft constraints. Hard constraints express all-or-none rules that restrict the values of a specific variable an item may have if it is to be included in a sample. For instance, *hard-bound constraints* allow the user to impose specific upper and lower bounds on the values of a variable for all the items in the sample. These constraints take precedence over the soft constraints and serve to filter items that can potentially be included in a sample.

In contrast, soft constraints do not operate on the basis of all-or-none rules; rather, they express all violations as matters of degree. For instance, a simple cost function for minimizing differences between two conditions $c1$ and $c2$ on a column of data containing each item's value on a variable, $\mathbf{x}$, is to square the difference between the means of the two conditions on that variable:

$$O_{MIN}(\mathbf{x}_{c1}, \mathbf{x}_{c2}) = (\overline{\mathbf{x}}_{c2} - \overline{\mathbf{x}}_{c1})^2. \quad (1)$$

Soft constraints are the main type of constraint that a user will employ during an optimization, and all have the same general form as the simple constraint outlined above. Many different soft constraints have been implemented to express a variety of desired relationships between the variables contained in one or more samples. These soft constraints can be subdivided into two main classes: simple constraints and meta-constraints. Simple constraints allow for (1) minimizing or maximizing differences in the means or variances between variables at either the group or the item level between two samples or within a sample (*two-sample distance constraints*), (2) matching means or variances to target values (*one-sample distance constraints*), (3) distributing values of variables evenly across the range of the population or the current sample (*soft entropy constraints*), which may be particularly useful in mixed-effects/regression designs, and (4) matching the correlation between two variables to a target value (*correlation-matching constraints*), which is often zero so as to eliminate correlations between independent variables that increase collinearity and weaken statistical power.

Meta-constraints operate on the same principles as simple constraints but serve to constrain the values of other constraints. This effectively allows the user to control the importance and degree to which different constraints are satisfied, which may be relevant in some optimizations. These constraints can be used (5) to allow a constraint that maximizes the difference between two conditions to be optimized only once another constraint that minimizes the difference between two conditions has been minimized—for instance, to ensure that differences on nuisance variables are eliminated before maximizing the differences between conditions on a variable of interest (*conditional matching constraint*)—or (6) to require that two constraints be satisfied to the same degree (*matching constraint*).

The full set of cost terms that operationalize the constraints a user imposes on item selection constitutes the total cost of the set of stimuli—that is, the degree to which all of the constraints have been satisfied. For instance, in an investigation of word frequency effects using both a low- and a high-frequency word condition, the full set of constraints might consist of a two-sample distance constraint that maximizes the differences between the samples on word frequency, such that the mean word frequency is lower in the low-frequency condition than in the high-frequency condition, and other two-sample distance constraints that minimize the mean groupwise differences on confounding variables such as word length.

3. Associate each sample with a population of items and fill that sample with a (random) sample from that population. This sample typically constitutes a very suboptimal sample, as expressed by a relatively high cost value (in comparison with other cost values obtained later in the optimization). Other initial sampling methods, such as specifying all or part of the items in a list, are discussed later in the article and in the user manual.
4. Attempt to improve the samples by randomly selecting two items, one from a particular sample and the other either from its population (*population feeder* item selection) and/or from another sample sharing the same population (*sample and population feeder* item selection), and swapping them. If the swap results in a reduction of the total cost, the optimization "moves" to the swap set; otherwise, the original samples are retained, and the optimization "stays" with the original set. Each of these attempted swaps is referred to as an *iteration* and can be accomplished in a computationally efficient manner via a *local update*. Local updates consist of calculating the difference in cost resulting from changing the two items, rather than recalculating each cost term using the whole data set. For example, rather than recomputing the mean by averaging the sum of values across all items, the value of the item being swapped out can be removed from the previously calculated mean, and the value of the item being swapped in can be added to that mean (see the user manual for details).
5. Statistically evaluate how well the constraints have been satisfied—for instance, by using *t*-tests to determine

whether differences across variables to be maximized have *p*-values less than .05 and differences that are to be minimized have *p*-values greater than .5. Other statistical tests are also available to evaluate whether a correlation between variables has been matched to a target value (often zero, to remove confounding covariates) or to evaluate whether the values of a variable are uniformly distributed (either across the range of values in the sample or across those in the population, depending on the intended generalizations to be drawn from the items). If these statistical criteria have been met (or no improvement in the value of the cost function has occurred for a specified number of iterations, called the *freeze point*), the optimization terminates.

6. Assess the degree to which the selected items represent the broader population. This is accomplished by running the optimization procedure several times and comparing the degree of overlap in the selected items across different runs. High overlap suggests that only a restricted subpopulation of items is capable of satisfying the constraints. Consequently, generalization of the results obtained with these items to the broader populations associated with each sample may not be appropriate. Conversely, low overlap suggests that results obtained with these items can be generalized to the broader population.

This greedy optimization routine is extremely rapid, by virtue of only making swaps that lower the overall cost of the samples, and may be sufficient to satisfy an experimenter's constraints in many cases. Additionally, the results of such an optimization are used to configure the more advanced stochastic optimization routine, as is discussed later. Consequently, a greedy optimization is prescribed as the first type of optimization to run. However, greedy optimization has a drawback in that making only swaps that decrease the cost may sometimes result in the algorithm becoming prematurely stuck in a *local minimum*, where swapping any one pair of items would result in a cost increase. Thus, despite the fact that a better *global minimum* cost value may exist elsewhere in the set, reaching it is not possible without at least some swaps that increase cost. The existence and likelihood of becoming stuck in such a local minimum is a function of several characteristics of a given optimization problem and is usually not explicitly determined, because this would require the use of a computationally impractical exhaustive search of all possible samples. As a general guideline, however, the likelihood of becoming trapped in local minima increases as sample size increases, as population size decreases, and as the number and complexity of the constraints increase (see Hinton & Sejnowski, 1986, for additional discussion).

Stochastic optimization

A principled means of avoiding these local minima is instantiated in the stochastic variation of the optimization search. Here, instead of applying a strict "move to the set of items with the lowest cost" rule, the cost difference, $\Delta cost$, between the original set of items and the swap set only biases the likelihood of choosing the set with the lower cost, with an additional term adding random variability to this choice as in classic stochastic relaxation searches (Hinton & Sejnowski, 1986; Kiefer & Wolfowitz, 1952; Rumelhart, Smolensky, McClelland, & Hinton, 1986). The variability in the decision to swap to a different set of items formally guarantees that an optimal set will be found if the optimization is run for a sufficient number of iterations, since it allows the algorithm to escape from local minima.

The amount of random variability can be manipulated via a parameter referred to as *temperature* (T), where higher temperatures increase the random element of the decision and where temperature has a lower bound of zero. The likelihood of swapping a pair of items is determined in a sigmoid function that depends on both the $\Delta cost$ and temperature parameters, as formalized in the following equation:

$$p_{swap}(\Delta cost, T) = \frac{1}{1 + e^{\frac{-\Delta cost}{T}}}.$$

Figure 1 illustrates this relationship for several sigmoid functions generated with different values of temperature and $\Delta cost$. As this figure shows, temperature modulates the degree to which $\Delta cost$ can bias the likelihood that a swap that reduces cost will take place: An infinitely high

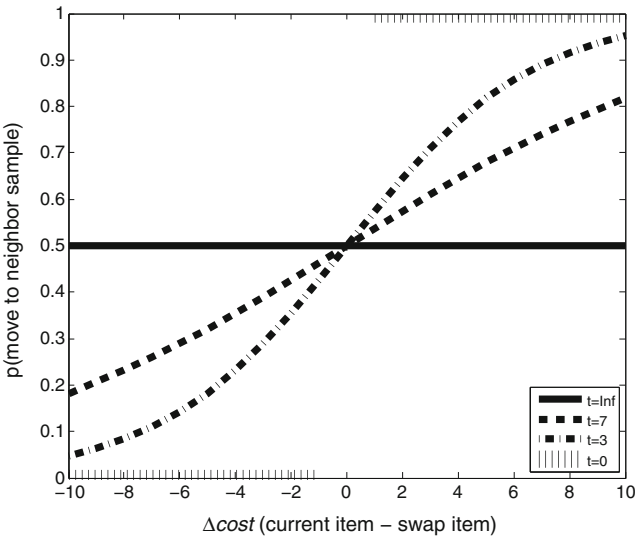

Fig. 1 Depiction of the likelihood of "moving" to the "swap" item or "staying" with the current item based on a range of values of the cost difference, $\Delta cost$, between the two items that could be swapped and on temperature. Inf = infinite

temperature causes swaps to be made without any consideration of whether the swap increases or decreases cost, whereas a temperature value of zero eliminates random variability and is equivalent to a greedy search. Consequently, fully leveraging the stochastic algorithm requires using an intermediate temperature value that ultimately leads to reductions in cost without causing the algorithm to become stuck in local minima.

Rather than attempting to identify a single temperature value, we have found, as others have before, that more rapid and reliable optimizations are possible by using an *annealing* function that gradually lowers temperature throughout the course of the optimization. Specifically, temperature decays at an exponential rate that has been shown to be optimal in related optimization problems (Ackley, Hinton, & Sejnowski, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983). The user manual and the example in Appendix 2 detail how this temperature-annealing schedule is calibrated on the basis of the results of the greedy optimization. Essentially, this process simply involves initially setting temperature to a sufficiently high value such that the vast majority of swaps are being biased by Δ*cost* only to a trivial degree. Subsequently, temperature is gradually lowered toward a Δ*cost* value that allows for a small percentage of swaps to still occur when an equivalent greedy optimization reaches its freeze point. This allows for the algorithm to escape from the minimum found in the greedy search if it is only a local one.

SOS performance on a range of optimization problems

We next report the results of Monte Carlo simulations on several example optimization problems to demonstrate the value and robustness of SOS. In the first of these examples, we examine a set of previously published psycholinguistic experiments for which the stimuli were manually selected to conform to a standard one-way design that dichotomizes a continuous variable (Morrison & Ellis, 1995). Using the same population of items these researchers had at their disposal, we compare the results of manual selection with those of SOS and show that SOS yields superior matches. Subsequently, we report how SOS can be used to construct another set of stimuli optimally suited for use in what is, in our view, a superior mixed-effect regression analogue of the original one-way designs used in two separate experiments.

After reporting the performance of SOS on these detailed examples, we briefly report the results of several additional realistic optimizations that correspond to experimental designs often encountered by researchers. Again, note that although all of the examples we present in this section concern psycholinguistic research questions, we cover designs that any psychological researcher might encounter, irrespective of his or her domain of inquiry. The scripts that contain the commands for each realistic case are available via the online user manual and may be useful starting points for users who want to create their own optimizations.

Comparison of SOS stimuli with manually selected stimuli in a two-level, one-way design

As a first illustration of the SOS algorithm, we elected to compare its performance with the items selected manually in an influential study by Morrison and Ellis (1995; hereafter, ME95). They examined whether the classic "word frequency" effects observed in a range of tasks (e.g., Schilling, Rayner, & Chumbley, 1998) have been overestimated because of the correlation between word frequency and another variable, the age at which words are learned, or "age of acquisition" (AoA; Gilhooly & Logie, 1982). To try to disentangle these effects, ME95 designed several experiments in which frequency and AoA were each varied independently while the other variable was held constant. As an example case for SOS, we will examine the stimuli ME95 designed to examine the effects of word frequency in isolation (Experiment 2): 24 pairs of high- and low-frequency words that differed on frequency while being matched on AoA, word length (in number of letters, expressed in the *letters* variable), and imageability.

On the basis of ME95's description of the way in which they selected stimuli, we recreated the population of items they had at their disposal by taking the union of the items having both AoA and imageability ratings (Gilhooly & Logie, 1980) and word frequency data (Kučera & Francis, 1967) in the MRC psycholinguistics database (Coltheart, 1981). In this population, the correlation between AoA and frequency was .22 and ranged from .04 to .67 between each of these two variables and the length and imageability variables. Next, following ME95, we created a high-frequency population from words with frequencies greater than 110 per million, and a low-frequency population from words that occurred 10 times per million or less. By manually selecting 24 pairs of stimuli from these two populations, ME95 were able to vary the high- and low-frequency stimulus lists significantly on frequency ($p = 2.5 \times 10^{-8}$ based on a two-tailed *t*-test). The two lists were matched perfectly on length ($p = 1.0$). However, the two lists were matched less well on AoA ($p = .20$) and not at all on imageability ($p = .01$).

With the aim of outperforming manual stimulus matching, we used SOS to select high- and low-frequency words from the same population as that used by ME95. A tutorial detailing the full process of running this optimization in the SOS software is included in Appendix 2. In brief, this process involved creating constraints indicating that the words in the low-frequency condition should have lower frequencies than the words in the high-frequency conditions, constraints indicating that the words in both conditions

should be matched on length, AoA, and imageability, and additional constraints that allowed for frequency differences to be maximized only once length, AoA, and imageability had been relatively well matched (see the user manual for additional motivating details for these constraints). We then dictated that the algorithm should continue to run until the two lists differed on frequency with a $p < .05$ and were matched on AoA, length, and imageability ($p < .5$). All tests and constraints were pairwise in nature, as in ME95.

Using an initial greedy optimization, we calibrated the parameters for the temperature-annealing function used in the stochastic optimization on the basis of the assumption that the greedy version had become prematurely stuck in a local minimum (as described above and in detail in Appendix 2). We then proceeded to run a stochastic version of the algorithm (referred to as the SOS optimization). This optimization found a solution that exceeded the statistical criteria we set: the high- and low-frequency samples that were selected varied on frequency [paired $t(23) = 16.13$, $p = 4.92 \times 10^{-14}$], but not on AoA [paired $t(23) = -0.68$, $p = .51$], length [paired $t(23) = 0.33$, $p = .75$], or imageability [paired $t(23) = 0.39$, $p = .70$]. These results show that SOS quickly found sets of stimuli that are better suited to test the theoretical question posed by ME95. In particular, the ME95 samples were less significantly different in terms of word frequency, while also being less well matched on AoA and imageability (which differed significantly across their two samples).

The descriptive statistics for the high- and low-frequency samples identified by ME95 and two SOS optimizations are presented in Table 1 and offer additional insight into how SOS obtained better matches (the SOS[length] optimization is described in detail later). This table shows that whereas both SOS optimizations generally found approximately equivalent or substantially better matches than did ME95 on AoA, length, and imageability, the mean difference in word frequency was, in fact, smaller in these samples than in ME95. Nevertheless, a more significant effect of word frequency resulted, because there was increased variability in the difference scores for the ME95 match; the effect size of the frequency difference was 1.7 for the ME95 matches versus 3.3 for the SOS optimization. This should translate into decreased within-condition error when behavioral data related to these items are analyzed and a reduced reliance on the data from the few items with the most extreme frequencies. Without going into detail here (but see Appendix 1), the cost function used by the pairwise minimization constraints also served to minimize any correlation between the two samples in ways that are not reflected in the *t*-tests but do manifest themselves in measures of collinearity, such as the variance inflation factor. For instance, the variance inflation factor for the frequency effect was 16% higher in the ME95 matches than in the SOS optimization, which would lead to more powerful frequency inferences if the matched variables were included as covariates. Taken together, these results serve to demonstrate how the constraints instantiated by SOS can lead to the discovery of alternative, superior item sets, sometimes in potentially unexpected ways.

An examination of Table 1 reveals a further characteristic of the optimization that is worth additional discussion: The high- and low-frequency samples discovered in the SOS optimization, although nonsignificantly different on length ($p = .51$), are not as well matched on the variable of length as those reported by ME95 ($p = 1.0$). This outcome is a natural effect of the SOS algorithm: All constraints will tend to be matched to the same level unless a particular constraint is prioritized. Insofar as there is no strong theoretical basis for prioritizing a particular constraint, such behavior is usually desirable, since it tends to eliminate all confounds to

Table 1 Descriptive statistics for the samples used in Experiment 2 of Morrison and Ellis (1995) and for the optimized samples identified by SOS

Sample		Frequency		AoA		Length		Imageability	
Low-frequency condition	ME95	6	(3)	3.58	(0.02)	5.04	(0.81)	4.62	(0.72)
	SOS	5	(3)	3.89	(0.91)	5.71	(1.73)	4.80	(0.85)
	SOS[length]	6	(2)	3.86	(0.83)	6.38	(1.56)	4.84	(0.78)
High-frequency condition	ME95	224	(129)	3.57	(0.83)	5.04	(0.81)	4.62	(0.92)
	SOS	153	(45)	3.86	(0.88)	5.75	(1.85)	4.81	(0.90)
	SOS[length]	161	(47)	3.85	(0.76)	6.38	(1.56)	4.85	(0.82)
(High − Low) frequency condition	ME95	218	(130)	−0.01	(0.01)	0.00	(0.00)	−0.47	(0.86)
	SOS	148	(45)	−0.03	(0.24)	0.04	(0.62)	0.02	(0.20)
	SOS[length]	157	(47)	−0.01	(0.18)	0.00	(0.00)	0.01	(0.21)

Standard deviations are in parentheses. ME95 = manually selected stimuli. SOS = stimuli selected by SOS as described in text and in Appendix 2. SOS[length] = stimuli selected by SOS with an additional weighting parameter to emphasize matching the conditions on word length. The (High − Low) Frequency section contains the mean pairwise difference and standard deviation of these difference scores across the two samples

approximately the same extent. However, if precise matches on one variable are particularly important for theoretical reasons, SOS can be configured to attempt to satisfy the constraints imposed on this variable to an increased degree. To illustrate this capacity, we ran an additional optimization in which we added a weight of 100 on the length constraint, as well as changed the statistical criterion for length to require a perfect match between the conditions (p = 1.0). This weighting parameter was chosen so that the initial cost values associated with length dominated overall cost. The choice of a value of 100 is not specific to this problem and generally produces this type of distribution across the cost terms, because SOS operates on the normalized values of each variable, as described in the user manual. Additionally, we modified the statistical criterion for the frequency test such that an equally significant difference in frequency was required to stop the optimization as was obtained in the human match ($p = 2.5 \times 10^{-8}$). This change was made because, otherwise, the more lenient p-value criterion of .05 was exceeded, and the optimization was stopped before much better sets were discovered. In this new optimization, we perfectly matched high- and low-frequency samples on length (p = 1.0), while still exceeding the original statistical criteria we stipulated and the matches obtained by ME95 (AoA, p = .81; imageability, p = .82; frequency, $p = 3.2 \times 10^{-14}$). For comparison purposes, Table 1 also lists the descriptive statistics for this additional set of items (the SOS[length] optimization).

In addition to evaluating the internal validity of the items produced by SOS during the course of the optimization, we also assessed the degree to which this optimization selected stimuli that were representative of the underlying population. As was described earlier, we cannot make claims about the generalizability of any experimental effects observed with these stimuli if they represent an idiosyncratic subset of the population. To evaluate the generalizability of the SOS stimuli, we ran five different stochastic optimizations until each passed the statistical criteria we had set. We then calculated the overlap in SOS to determine how similar the resulting lists in each condition were to each other. On average, the high-frequency lists shared 13% (SD = 0.09) of the total number of items across each optimization, while the low-frequency lists shared 8% (SD = 0.06; values of 25% and 8% were obtained in analogous comparisons for the items discovered in the SOS[length] optimization). These values suggest that the five optimal solutions were, in fact, quite different from one another and that the algorithm was not discovering the same unique solution each time. Thus, we would expect any behavioral effects observed with these stimuli to generalize to new sets of stimuli on the basis of the principles of statistical inference. Specifically, many different items can be chosen to satisfy the specified constraints. Using any one of these samples thus corresponds to using one of many possible random samples of a broad population of items. Consequently, the results obtained with any one of these samples are likely to generalize to the population. Nevertheless, it is worth noting that generalization based on statistical inference alone can occasionally be misleading if the random sample that is selected is unusual in an idiosyncratic manner purely by chance. The most confident generalizations would thus also be supported by running more than one experiment, using different random samples of items that satisfy the experimental constraints. By evaluating the overlap between several sets of items generated by SOS, sets of items suitable for this task are already available for this purpose.

A regression analogue and expansion of the one-way design used by Morrison and Ellis (1995)

In the original ME95 work, the authors examined the effects of word frequency while holding AoA constant in one experiment (Experiment 2) and of AoA while holding word frequency constant in another experiment (Experiment 1). These inferences could also have been made in a single experiment using a standard 2 × 2 factorial design that crosses frequency and AoA. Such a design would carry the benefit of ensuring that the significance of a given effect was ascertained in the context of the same levels of the other effect. This is because the same mean AoA and frequency values would be used in all of the statistical comparisons and this need not be the case across different independent experiments, which could confound the results. Practically, however, using such a design is not possible on the basis of how the conditions were defined in ME95; the hard cutoffs for inclusion in the high- and low-frequency and in the early- and late-AoA conditions used in Experiments 1 and 2 do not leave enough items in the four cells of the proposed design to create samples of the same size as in the ME95 experiments, even before stimulus optimization. This also suggests that somewhat different populations of items were used to study each type of effect.

The inability to simultaneously study the effects of AoA and word frequency may, however, be a result of the restrictiveness of a factorial design. Specifically, it is often the case that researchers will impose hard bounds on the items that can enter into particular cells of the design so as to maximize the differences observed between conditions. Presumably, this is part of what motivated ME95 to employ the low-frequency upper bound of 10 and the high-frequency lower bound of 110 in their second experiment and to employ similar constraints on the early- and late-AoA samples in their first experiment. This strategy is problematic for several reasons. First, sampling items in this manner allows only items from the extreme ends of the continuous variables under study to be selected for use in the experiment. Interpolation to this range using a

standard linearity assumption, as in many analyses in psychology, is questionable in this case, since there are no data with which to evaluate this assumption. Second, finding items to fill the "atypical" cells in the design (e.g., the high-frequency, later-acquired words, which are rare given the negative correlation between frequency and AoA) is often difficult—if not outright impossible—because of the relatively small populations of items that can be considered for inclusion in these cells. Third, these designs can often be underpowered because within-condition variance on the variable of interest is transferred into the error term of standard statistical analyses. For example, words with frequencies of 110 and 11,000 would both be grouped together in the high-frequency condition, and this more precise frequency information would not enter into the analyses.

An alternative design, which does not suffer from these problems, is to sample a broad range of items that can be used to study the target effects via mixed-effects/regression analyses (Baayen et al., 2008). To do so, we used SOS to select 100 items with a uniform distribution over the range of AoA and word frequency values in a slightly restricted version of the population used in our first example.[2] This was accomplished using entropy constraints. We also included constraints that eliminated the correlations between the two main variables of interest, as well as between each main variable, length, and imageability, so as to minimize collinearity and boost the power of the regression. Statistical criteria were selected such that the optimization would terminate when all of the correlations were nonsignificantly different from zero ($p > .5$) and when the distributions were nonsignificantly different from uniform distributions ($p = .001$; Appendix 1 discusses the motivation for using a smaller p-value when testing the uniformity of a distribution).

Initial runs of SOS showed that the statistical tests related to the correlations were being passed but that the optimization terminated by reaching the freezing point when the frequency and AoA distributions still differed from uniform distributions (e.g., frequency was still positively skewed, with few high-frequency items in the sample). Alternatively, if only the distribution constraints and statistical tests related to frequency and AoA were included, the algorithm successfully found a satisfactory solution (albeit one with significant intercorrelations among the different variables). Which of these solutions is most desirable is debatable, since each reflects a different trade-off of internal and external validity and statistical power. Moreover, these two results suggest, as would be expected on the basis of the original ME95 experiment structure, that pulling apart these two dimensions is a nontrivial task. Nevertheless, we attempted to find a compromise between these two extremes by adding a weight to each of the frequency and AoA distribution constraints so that they were of the same or a larger order of magnitude as those of the correlation constraints at the beginning of the optimization; a weight of 100 satisfied this goal.

An initial Monte Carlo simulation showed that the updated version of the greedy optimization succeeded in satisfying the statistical criteria 8/10 times. Using the results of a randomly selected failed optimization, we then configured a stochastic version of this same simulation. In this case, the optimization succeeded 10/10 times. To illustrate the effect of the entropy constraints on creating uniform distributions, the initial and final AoA and frequency distributions are presented in Fig. 2.

Additionally, we computed the overlap between the 10 samples that were discovered by the stochastic optimization. The overlap between these samples was 42%, suggesting that the satisfaction of this constraint does depend in part on a subset of these items. Consequently, a researcher might wish to consider relaxing some of the statistical criteria if the external validity of the experiment is paramount.

Performance of SOS on additional examples

Next, we briefly report the results of several realistic optimizations corresponding to additional experimental designs often encountered by researchers. These optimizations demonstrate the capabilities of SOS in a variety of other real-world scenarios. Our examples are based primarily on the word data from the MRC Psycholinguistic Database (Coltheart, 1981) that have values for Kučera and Francis (1967) frequency and the number of letters, phonemes, and syllables for each word, unless otherwise specified. Variables were considered matched if $p > .5$ and significantly different when $p < .05$. All examples below were run as stochastic optimizations unless otherwise stated, and the number of successful optimizations (out of 10) is also included. The script files for these and additional designs are available in the user manual.

Categorical/ANOVA designs

One-way designs First, we implemented several one-way designs using two samples with 100 words in each sample. These designs are similar to the ME95 case described above (e.g., matching two lists for length, number of syllables, and number of phonemes, while maximizing frequency differences) but did not involve the stipulation of upper and lower bounds for inclusion in the two different samples. Instead, SOS discovered the optimal separation between these distributions while

[2] SOS was unable to create a uniform distribution over the full range of the frequency variable in the initial population of items, even when this was the only constraint in the optimization. This was because the frequency data in the population were strongly right-skewed and there were insufficient high-frequency items to sample the upper end of the frequency continuum uniformly. To avoid this problem, the top 5% of words with the highest frequencies were trimmed.

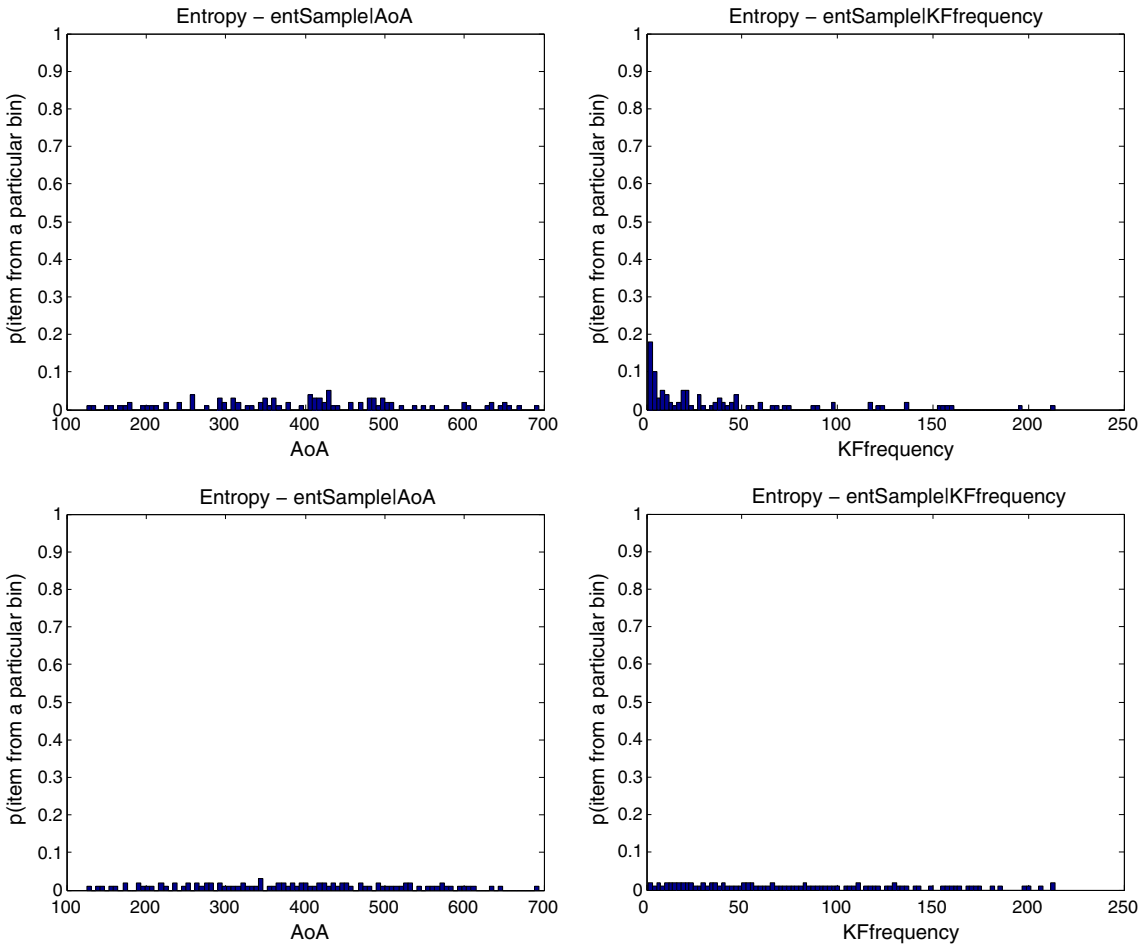

Fig. 2 Plots generated by the SOS software of the pre- and post-optimization AoA and frequency distributions when a constraint was imposed to pressure the creation of uniform distributions across the populations (AoA range, 125-694; frequency range, 1-214). The preoptimization AoA and frequency distributions had entropy values of −0.012 and −0.071, and these entropy values increased to −0.0095 and −0.0061 by the end of the optimization. These final distributions did not differ from a uniform distribution ($p > .001$)

sampling all items from the same population. These optimizations succeeded 10/10 times when matched groupwise.

A similar optimization also succeeded 10/10 times when items were matched pairwise, with the following caveat. Because of the details of the cost function used to minimize pairwise differences between conditions, pairwise matching also reduces the variance in the pairwise differences. This sometimes produces samples that, while differing only very slightly in mean values, yield *p*-values less than .5 for variables intended to be matched (see Appendix 1 for additional discussion). In one instance of this optimization, for example, the mean number of phonemes in the high-frequency sample was 5.08 (SE = 0.17), while the mean number of phonemes in the low-frequency sample was 5.02 (SE = 0.18). Although a difference of 0.06 phonemes is very unlikely to affect the behavior of participants in a substantial manner, the low variance in each list caused the samples to be under the $p > .5$ cutoff for matching constraints, t[paired] (99) = 1.23, p = .22. To prevent these effectively "good" samples from being rejected by SOS, we have implemented a "threshold" parameter that allows statistical tests in SOS to be passed if the mean difference between two samples is less than a specified amount. In the example discussed here—two samples maximized for frequency differences but matched pairwise on length, phonemes, and syllables—we dictated that optimizations could pass if the variables to be matched had means within 0.5 of each other. Depending on the problem at hand and the scale of the variables, users may opt to not use thresholding at all or to change the cutoff to more appropriate values for the particular samples they are attempting to create.

We also examined the performance of SOS when three, rather than two, final samples were desired; in other words, the experimental design in this case corresponds to a one-way design with three levels of frequency (low, medium, and high). Additionally, all of these conditions were matched for length, number of syllables, and number of phonemes. The three levels of frequency were not stipulated in advance. Rather, SOS discovered an optimal parcellation along the frequency variable, in which each level differed

significantly from the others, and the intermediate level was approximately equidistant between the low and high levels. This optimization produced acceptable stimuli 10/10 times matching items group-wise and 10/10 times matching items pairwise (a threshold of 0.1 was used when the samples were matched pairwise).

Two-way designs Often, researchers want to examine the influence of two variables simultaneously. These 2 × 2 designs often pose substantial difficulties for manual selection heuristics, especially when each cell in the 2 × 2 design also needs to be matched for multiple confounding variables (as in the ME95 example). Nevertheless, provided that sufficient items exist in the population to create such a design, SOS is capable of discovering appropriate samples. To demonstrate this, we had SOS create a 2 × 2 design that crossed the effect of word frequency with that of word length, while minimizing any differences on imageability across these four samples. Items were drawn from a single population containing all of the words in MRC for which length and AoA data were available and for which word frequencies were constrained to be less than or equal to 1,000. This last restriction was imposed for the same reasons as described in the regression analogue to the ME95 design. No restriction on which type of item could be placed into a high or low cell on a particular variable was imposed such that the optimizer was required to discover an optimal division between the conditions. SOS found samples that satisfied these constraints 10/10 times with both groupwise and pairwise constraints, although threshold parameters were required in the pairwise version of the optimization.

Other optimization problems

Another feature of SOS is the ability to specify some stimuli as being "fixed." This allows SOS to discover a sample that possesses certain relations to a predetermined list—for instance, items from a previous experiment that is being extended. For example, we asked SOS to find a sample of 100 nouns that were matched to 100 preselected verbs for frequency, length, and number of syllables and phonemes. This optimization was successful 10/10 times when matched groupwise and 10/10 when matched pairwise (a threshold of 0.1 was used when the samples were matched pairwise). In a similar vein, SOS can also be used to find two samples of words matched to specific, user-inputted values of frequency while still matching these samples on length, number of syllables, and number of phonemes. This optimization was successful 10/10 times groupwise and 10/10 times pairwise.

Additional types of optimizations, including but not limited to matching the variability across conditions, matching to a target correlation, spreading scores uniformly within a sample rather than across the range of the population, and replacing only a subset of items in a sample, are also possible in SOS (see Appendix 1 and the user manual for details).

Extended applications

Although the discussion to date has focused primarily on the selection of stimuli for new psycholinguistic experiments, SOS can be applied to a wide range of optimization problems. Moving beyond stimulus selection, one exciting potential application of SOS is in *participant* selection, particularly in the context of investigating the effects of a neurological dysfunction such as semantic dementia, herpes simplex encephalitis, or stroke on a cognitive system such as semantic memory. Patterson and Plaut (2009) have recently argued that case-series analyses (e.g., Woollams, Lambon Ralph, Plaut, & Patterson, 2007) may be more useful than single-case studies when studying the mechanisms that underlie these systems and their neural substrates. Conducting such case-series studies, however, is still quite a challenging endeavor because of the need to match individual patients with healthy controls on a number of variables (e.g., age, years of education, performance on baseline intelligence tests). Using SOS to select these controls could greatly enhance a study's ability to inform researchers on the underpinnings of a particular type of impairment.

Another application of SOS concerns the reanalysis of a subset of items from an experiment that was found, post hoc, to use a suboptimal set of items. For instance, Watson (2009) matched two categories of verbs, those with strong or weak associations with specific motor programs, on published imageability ratings. Later, she discovered that the published verb ratings did not correlate with ratings elicited by participants when verbs were rated within their grammatical context (e.g., "the race" vs. "to race"). Moreover, a regression-based solution to this problem was suspect because the ratings were bimodally distributed. By selecting an optimal subsample of items with SOS (treating the existing sample items as the population), Watson was able to reanalyze these data without the presence of any imageability confounds. SOS was also successfully used for a similar problem where population and/or dialect differences led to a confound on familiarity measures in the items used by Armstrong and Plaut (2008) and in selecting a larger set of items for use in follow-up work (Armstrong & Plaut, 2011).

Another use of SOS has been to leverage data from behavioral piloting for functional magnetic resonance imaging (fMRI) experiments to preemptively select ideal stimulus sets. With standard block-design fMRI experiments, it is often desirable to have different conditions (blocks) equated

for accuracy and reaction time, ensuring that differences in neural activity are not due to difficulty confounds (Binder, Desai, Graves, & Conant, 2009). Frequently, researchers collect behavioral data on the tasks that will be presented in the scanner, and, if accuracy or reaction time differences exist, they are forced either to manually select subsets of blocks or individual trials that are equated on these measures or to include these measures as covariates in analyses of the BOLD data itself. When Watson and Chatterjee (2012) encountered this issue, they instead used SOS to select items from the behavioral pilot experiment that were equated for difficulty across conditions.

Using and modifying SOS

One guiding principle in the development of the SOS algorithm and software was that it would be impossible to anticipate all of the potential uses and insights other researchers might have with regards to the optimization procedure. The SOS software was therefore written in a standard, multiplatform, easy-to-program environment (MATLAB) that is already familiar to many psychologists. The source code for the software has been made freely available for academic purposes. The use of object-oriented programming practices, combined with the implemented examples of different constraints and cost functions discussed in this article, the user manual, and the source code documentation, should also facilitate the modification and extension of this code for new purposes.

Conclusion

Despite the critical importance of selecting optimal stimuli, this aspect of experimental design has largely been ignored, while other aspects have benefited from massive advances. In the present work, we have outlined some of the obvious and not-so-obvious effects that suboptimal stimuli can have on the internal and external validity of experiments and, ultimately, on the overall value of a particular study. We have also provided a theoretical basis and empirical support for a solution to these problems—the SOS algorithm and software—which implements an optimization tailored for easy use in psychological research. This tool allows researchers to focus on choosing an optimization problem that suits their theoretical question, rather than on the tedious task of manually selecting stimuli and the inherent limitations of that method. We anticipate that many researchers will find this tool useful when selecting experimental stimuli, and, looking ahead, we expect SOS will be further developed and extended to enhance many other aspects of experimental and analytical procedures.

Acknowledgements This research was supported by an NSERC Alexander Graham Bell Canada Graduate Scholarship to B.C.A., an NIH training grant (T32 NS054575-04) to C.E.W. as a trainee, and a Commonwealth Universal Research Enhancement Program Grant to Carnegie Mellon University and the University of Pittsburgh. We thank Natasha Tokowicz, Sarah Laszlo, Yevdokiya Yermolayeva, and the members of the Reading and Language group at the University of Pittsburgh for helpful comments and discussion. An early draft of this work was included in C.E.W.'s doctoral dissertation. Correspondence about this article may be sent to blairarm@andrew.cmu.edu or watsonc@mail.med.upenn.edu. Correspondance about the SOS software may be sent to sos@cnbc.cmu.edu.

Appendix 1

Implemented cost functions

A number of constraints and associated cost functions have currently been implemented in the SOS software and can broadly be divided into two categories: hard constraints and soft constraints. Hard constraints effectively amount to strict inclusion/exclusion rules governing the values of a variable in a condition and are functionally equivalent to filtering out items that exceed specified cutoffs from the population before beginning to optimize the stimuli. For instance, a hard constraint can be used to specify that all of the word frequencies in a low-frequency condition must be below a particular value. In contrast, soft constraints have continuous cost functions that denote the degree to which a set of stimuli satisfies the constraints. In principle, all possible sets of stimuli could be selected to satisfy the constraints to greater or lesser extents.

Because the interesting and practically useful aspects of the stochastic optimization algorithm relate to maximally satisfying soft constraints—particularly of the "simple" type, introduced in the next section—the main focus of our work has been to enrich the vocabulary of these soft constraints for users to employ in their own optimizations. We therefore discuss these soft constraints first and return to the issue of hard constraints later.

Soft constraints

Simple soft constraints Simple soft constraints are soft constraints that directly reflect differences in normalized values on specified variables. These are the main type of constraint that needs to be stipulated before beginning the optimization. There are four main types of simple soft constraints: single-sample distance constraints, two-sample distance constraints, entropy (uniformity) constraints, and correlation-matching constraints. We begin by introducing two-sample soft distance

constraints because they are likely the most widely applicable type of constraint.

Two-sample distance constraints This first type of constraint is concerned with minimizing or maximizing the distance between two samples of data on a particular measure, $f(\mathbf{x})$, at either the group or the item level. Currently, this distance measure corresponds to either the mean or the standard deviation of each sample on a particular variable. In most applications, the data are expected to correspond to the values of the same variable across two different conditions (e.g., maximize the difference in word frequency across a low- and a high-frequency word condition). This notwithstanding, the software implementation also allows for the data from two different variables to be compared within a single condition or for two different variables to be compared across different conditions. For simplicity of notation, we will assume that the cost function is measuring the same variable across two different conditions in the following discussion.

As was mentioned previously, maximizing or minimizing the distance between two samples on a particular measure, applied to the item data for each sample $f(\mathbf{x})$, can be operationalized using the simple cost penalty function O_{MIN} $(\mathbf{x}_{c1},\mathbf{x}_{c2})$ in Eq. 1. In our experience using SOS, this function has proven to be a reasonable starting point for operationalizing the desired relationship between the two variables on $f(\mathbf{x})$. However, in some instances, it may be desirable to prioritize the importance of some constraints over others. To this end, a weighting coefficient, b (with a default value of 1.0), and an exponent, n (with a default value of 2.0), have been added as free parameters. To accommodate the usage of exponents, the absolute value of the difference between the two measurements must now also be taken so that all differences produce a positive value. The result is a more general minimization function for group-level differences:

$$O_{MIN}(\mathbf{x}_{c1}, \mathbf{x}_{c2}, b, n) = b(|f(\mathbf{x}_{c2}) - f(\mathbf{x}_{c1})|)^n. \quad (2)$$

A maximization function for group-level differences can be generated in a similar manner. A particular ordering of conditions can be enforced by adding the additional sign function $s(f(\mathbf{x}_{c1}), f(\mathbf{x}_{c2}))$ that equals 1 when $c1$ is less than $c2$ and −1 otherwise:

$$O_{orderMAX}(\mathbf{x}_{c1}, \mathbf{x}_{c2}, b, n) = -s(f(\mathbf{x}_{c1}), f(\mathbf{x}_{c2}))\, O_{MIN}(\mathbf{x}_{c1}, \mathbf{x}_{c2}, b, n). \quad (3)$$

Manipulating either the weight or the exponent allows for the prioritization of each cost function relative to the total cost. As a general guideline, small manipulations of the weight allow for more subtle emphasis of some constraints over others, whereas manipulations of the exponent induce more heavy-handed changes in cost. Weight manipulations are reminiscent of the β coefficients in a linear regression. This allows a researcher to prioritize the importance of satisfying constraints imposed on different variables on the basis of existing regression analyses that denote the degree to which each variable influences performance.[3] Exponent manipulations can be used to quickly increase or decrease cost to well above or below the level of other constraints. Such a manipulation effectively causes these constraints to be satisfied first before satisfying the other constraints. The details of how this occurs are discussed in the section on temperature annealing in the main text and in the user manual.

In addition to satisfying group-level constraints, similar constraints can also be imposed at the item level, provided there are equal numbers of items in the two conditions being compared. These pairwise constraints provide a tighter restriction on the relationships that exist across the items in the conditions being compared at the expense of being harder to satisfy. Pairwise constraints may be particularly beneficial in two situations. The first is, unsurprisingly, when a researcher desires to run pairwise statistical tests across items, thereby increasing the power of the experiment. The second situation occurs when, after matching a variable across conditions, any remaining variance due to the variable is to be extracted from the dependent variable using a regression-based analysis (e.g., multiple/mixed-effects regression, ANCOVA). In this case, controlling for a group-level measurement of the variable, such as the mean, may still lead to different distributions of that variable across the conditions. For example, imagine trying to match on word length at a groupwise level when creating high- and low-frequency conditions in a factorial design. Even by successfully matching the mean word length across the two conditions, it is still possible that the distribution of word lengths differs between them. For instance, word length might be skewed positively in the low-frequency condition and negatively in the high-frequency condition. When these two distributions are merged into a single "length" variable and are regressed out, the composite length distribution may be

[3] The relationship between β in regression and b in the cost functions is similar, but nonidentical, due to the nonlinearities in the cost and swap likelihood functions, although the two behave similarly. The similarity of this relationship is particularly strong in the context of multiple minimization constraints all being satisfied to similar degrees and when $\Delta cost$ falls within the relatively linear range of the sigmoidal p (swap) function.

correlated with the low- and high-frequency conditions.[4] Such a situation can lead to an interaction among the independent variables and a reduction in the variance in the dependent measure that can be uniquely ascribed to the effects of either length or frequency in the statistical analyses. Relatedly, this may enlarge the estimates of the variance for each predictor (i.e., increase the variance inflation factor). In contrast, if the distributions across the two conditions were matched on word length, as would occur if pairwise matching across stimuli is employed, such an interaction would not be possible. This can increase the amount of unique variance that is ascribed to the effect of word frequency and boost the statistical power of that effect. This is demonstrated in the regression-analogue example case in the main text.

Pairwise constraints require only a simple extension of the groupwise constraints presented earlier. The only difference between the two is that instead of calculating the particular measure of interest (e.g., mean) once across each condition before subtracting the measure of one condition from the other, the values for each *pair* of items, *i*, are subtracted from one another before calculating the measure. The absolute value of measures for each of the pairs can then be summed across all pairs, *k*, and the average measure can be extracted by dividing this sum by the number of pairs. For instance, to minimize the difference across all *k* pairs of items in two different conditions, the following cost function is used:

$$O_{pairMIN}(\mathbf{x}_{c1}, \mathbf{x}_{c2}, b, n) = \frac{b}{k}\left(\sum_{i=1}^{k} \left| f\left(\mathbf{x}_{c2,i} - \mathbf{x}_{c1,i}\right)\right|\right)^{n}. \quad (4)$$

The same basic adjustments can be applied to the cost function used to maximize group-level differences, yielding a pairwise maximization cost function.

Single-sample distance constraints A simplified variant of the two-sample soft distance constraint has also been developed that allows differences on a particular measure to be assessed relative to a user-specified value, *v*, instead of against the value of that measure in another sample. This type of constraint may be useful when generating a condition with a particular value on a particular variable or when minimizing the variability in a condition by matching the standard deviation of the condition to zero. The equations used to express the costs associated with these single-sample distance constraints are identical to those for the two-sample constraints, only the values of the data points in what we have labeled $\mathbf{x}_{c1}$ are replaced with the user-specified value. For instance, to minimize the group-level difference between a condition, $\mathbf{x}_{c2}$ and a target value, *v*, we simply substitute *v* for $f(\mathbf{x}_{c1})$ in Eq. 2:

$$O_{valueMIN}(v, \mathbf{x}_{c2}, b, n) = b(|f(\mathbf{x}_{c2}) - v|)^{n}. \quad (5)$$

The same change can be applied to all of the other two-sample distance constraints discussed previously.

Soft entropy constraints By themselves, soft distance constraints are often sufficient to stipulate the types of relationships among stimuli that are imposed in standard multilevel or factorial designs. However, Baayen et al. (2008) recently criticized these types of factorial designs and argued that they are ill-suited for studying the effects of continuous variables (e.g., studying word frequency effects using low- and high-frequency conditions).

There are two reasons why these designs are problematic. First, they often employ distinct collections of items that are as widely separated as possible on the variable of interest so as to maximize statistical power. Insofar as the distributions across conditions do not overlap, this weakens the degree to which the effects observed at these extreme ends of the continuum can be interpolated to the intermediate values between the conditions that have not actually been sampled. Indeed, strict adherence to statistical theory dictates that statistical inferences cannot be made to these items, which were not part of the effective populations that were sampled to create the conditions. Moreover, even if this restriction were relaxed, this type of design would require the strong and sometimes questionable assumption that these extremes of the continuum are linked by a particular (usually linear) function that cannot be empirically verified with the items used in this design [e.g., Balota, Cortese, Sergent-Marshall, Spieler, & Yap, 2004, and Brysbaert & New, 2009, report a nonlinear effect of log(frequency) for high-frequency words that would be incorrectly modeled by a linear function]. Second, these types of designs can often be statistically underpowered because they discard the exact data points associated with individual items—represented on a continuum—by collapsing them into qualitative groups in which all items are treated identically and items across groups are treated differently. For instance, two words with frequencies of 1 and 50 might both be grouped together into a low-frequency condition, and two items with frequencies of 49 and 51 might be split into the low- and high-frequency conditions. To the extent that more precise data can actually account for more variance, this kind of falsely dichotomous design results in a loss of statistical power.

[4] Note that such a relationship may be present even if a pairwise *t*-test across the values in each condition produces a *p*-value of 1.0. This is because the *t*-test assesses whether the mean deviation across pairs—as opposed to the sum of the absolute values of the deviations across pairs—equals zero. A *t*-test *p*-value of 1.0 is therefore not sufficient to guarantee that this problem has been avoided.

After considering these issues, Baayen et al. (2008) concluded that a superior experimental design would involve sampling items that span the whole continuum of values of a given variable and using a form of multiple regression to analyze the results. This procedure has the effect of boosting both the theoretical validity and statistical power of the analyses. The challenge with a regression approach is that variables that are not uniformly distributed will be poorly sampled for some ranges of values and oversampled for others. As a result, statistical analyses can be biased and prevent accurate inferences from being made to at least a subset of the items. Such a scenario is particularly problematic when practical considerations, such as studying patient populations, permit only a relatively small number of items to be tested.

To take full advantage of the statistical framework advanced by Baayen et al. (2008), it is therefore useful to sample uniformly across the range of values for the variables that will be analyzed. The entropy constraint applies a pressure for stimuli to be distributed in this fashion, either across the range of values for that variable in the entire population or only within the sample. This allows a researcher to differentially accentuate either the external or the internal validity of the experiment, respectively.

The entropy constraint assesses uniformity by examining the degree to which the values of a variable, x, in a given condition are equally distributed in a histogram. The x-axis of the histogram is divided into n bins, where n is by default the number of items and has lower and upper bounds corresponding to the lower and upper bounds of the population or sample, as specified by the user on the basis of the scope of the desired generalization. The y-axis denotes the frequencies of items in the condition that fall within a given bin. An example of such histograms as generated by SOS is presented in the regression-analogue example case in the main text. The entropy, or uniformity, of this distribution can then be formalized as a function that increases in value as the distribution of the variable becomes more uniform. One function that satisfies this criterion is to define a simple measure of entropy, S_{simple} as a function of the proportion of items, p, in bin i of the histogram, summed across all n bins:

$$S_{simple}(p) = -\sum_{i=1}^{n} p_i \ln p_i.$$

This equation amounts to a simplified version of Boltzmann–Gibbs entropy in which a constant has been set to 1 (Salazar, Plastino, & Toral, 2000). In this case, entropy is maximized when the values of p_i are identical for all i—that is, when values are uniformly distributed in the histogram.[5] This simple entropy equation is then subjected to some minor algebraic adjustments to bring it in line with the other cost functions that have been presented. First, the data on which the other constraints operated were normalized before cost was calculated so that all constraints would make similar contributions both within and across different optimizations. As it stands, this is not true for the entropy formula we have introduced, whose upper and lower bounds change depending on the number of items that are included in the sample. The first step in correcting this problem is to ensure that the entropy function can produce a defined value for every possible distribution of values. However, when the proportion of the total number of items in a bin is zero, the result is undefined. Our solution to this problem was to simply add a constant of one when calculating the natural logarithm. Because the natural logarithm produces positive, as opposed to negative, values when applied to numbers greater than one, the resulting equation now produces negative values, but larger (closer to zero) values still reflect a more uniform distribution, as before. Furthermore, the entropy equation can be standardized so that different sample sizes with the same overall distributions of stimuli produce identical entropy values. This is accomplished by dividing the previous equation by the number of items in the condition:

$$S'_{\text{simple}}(p) = -\frac{\sum_{i=1}^{n} p_i \ln(p_i + 1)}{n}.$$

Finally, to make the entropy equation more intuitive, it is useful to standardize its range to an interval such as [−1,0]. Such a normalization would facilitate the interpretation of intermediate values and make the boundary values for the equation more evident. To accomplish this, it is first necessary to calculate the upper and lower bounds of entropy. The lower bound corresponds to the case where all stimuli fall within a single bin and all of the other bins are empty. In this case, the contributions to entropy for all of the empty bins will equal zero (because $\ln(1) = 0$). Substituting into the existing entropy formula, the lower bound therefore simply corresponds to $\frac{\ln(2)}{n}$. Similarly, the upper bound corresponds to the case where each bin contains 1 of the n stimuli. Substituting into the entropy formula, the upper bound is, therefore, $\frac{1}{n}\ln\left(\frac{1}{n} + 1\right)$, where the summation across n and the division by n have canceled each other out. Having established the upper and lower bounds, the entropy formula can then be standardized to have a range of [–1,0] via a

[5] In strict terms, there is no attempt to distribute the data uniformly within each bin, so this is true only in the details with large numbers of bins/items. Other, more sophisticated methods could be employed to ensure a perfectly uniform distribution, but we do not believe that their complexity and computational overhead are justifiable in the vast majority of cases.

linear transformation. The standardization moves the upper bound to zero and divides by the difference between the upper and lower bounds, causing the lower bound value to be shifted to –1:

$$S(p) = -\frac{-1\ln\left(\frac{1}{n}+1\right) + \sum_{i=1}^{n} p_i \ln(p_i+1)}{\ln(2) - 1\ln\left(\frac{1}{n}+1\right)}.$$

The cost penalty used to maximize the entropy of a variable is then simply a function of the absolute value of entropy, so that higher entropy values generate lower costs:

$$O_{entMAX}(p, b, n) = b(|S(p)|)^n. \quad (6)$$

Soft correlation-matching constraints Another important consideration when constructing stimuli for multiple/mixed-effect regression analyses is the correlation that may exist between the independent variables. Typically, such a correlation is not desirable, because it reduces the degree to which variance accounted for by the overall model can be uniquely attributed to a particular independent variable. This causes an underestimation of the statistical significance of each of the independent variables (by increasing the error variance inflation factor) and, as the collinearity of the independent variables increases, can invalidate the model as a whole (because the coefficient matrix becomes singular; Healy, 1968). Consequently, it is typically desirable to minimize the correlations that exist between the independent variables. In particular, this may be especially useful when recasting classic factorial designs in terms of regression (Baayen et al., 2008), since this allows for continuous variables—ideally, distributed uniformally across a range, as is possible using an entropy maximization constraint—to be decorrelated without necessitating the creation of full-factorial models where all cells must be included to eliminate such correlations. For instance, such a correlation-matching constraint could be used to stipulate that the length and the frequency values for a given sample must be uncorrelated when creating an analogue of a 2 × 2 factorial design involving those variables, as in the second example case in the main text. Of course, classic factorial designs may further benefit from eliminating such correlations across additional variables that are included as covariates and not as separate factors in the design. More generally, it may also be desirable to match the correlation between two independent variables to a particular value—for instance, to show that an effect reported in a previous study is replicated when such a correlation is present but eliminated when the correlation is removed.

The correlation-matching constraint provides support for these types of scenarios. It allows for users to match the correlation between two variables, $r(\mathbf{x}_{c1}, \mathbf{x}_{c2})$ either within or between two samples, to a prespecified value, v. The cost function used to express this constraint is similar in form to that for the distance minimization constraints, with the addition of two modifying equations that alter the behavior at the boundary conditions when the variance in $\mathbf{x}_{c1}$ and/or $\mathbf{x}_{c2}$ is zero:

$$O_{matchCorrel}(\mathbf{x}_{c1}, \mathbf{x}_{c2}, b, n, v) = \begin{cases} b(|r(\mathbf{x}_{c1}, \mathbf{x}_{c2}) - v|)^n, & \text{if } \sigma_{\mathbf{x}_{c1}} > 0 \text{ and } \sigma_{\mathbf{x}_{c2}} > 0 \\ 1, & \text{if } \sigma_{\mathbf{x}_{c1}} = 0 \text{ and } \sigma_{\mathbf{x}_{c2}} = 0 \\ 0, & \text{if } \sigma_{\mathbf{x}_{c1}} = 0 \text{ xor } \sigma_{\mathbf{x}_{c2}} = 0. \end{cases} \quad (7)$$

Assuming the default values for the weight and exponent, the resulting function is bounded by [0,2] but usually operates between 0 and 1 when attempting to eliminate the correlation between two variables. This is particularly useful because the range of the cost associated with the correlation matching constraint is therefore quite similar to that of the entropy constraint. Consequently, when both constraints are used simultaneously, which is often likely to be the case, they should each be approximately equally satisfied by default.

Soft meta-constraints In contrast to soft simple constraints, which operate directly on measurements associated with the stimuli in a condition, soft meta-constraints serve to constrain the differences that exist between constraints themselves. Meta-constraints are useful in some instances when combinations of soft constraints and their respective costs do not correctly express the optimal pattern of relationships that should exist among the conditions. These meta-constraints also serve as a standardized means of reprioritizing the relative importance of other constraints that could otherwise require more fine-tuning by the user. Currently, two types of meta-constraints have been implemented: cost-matching meta-constraints and conditional cost-matching meta-constraints.

Cost-matching meta-constraint The first type of meta-constraint bears a strong similarity to the simple soft constraint used to minimize group-level differences on a particular measure. The main difference is that the measurements contributing to the meta-constraint have been substituted with the costs associated with two other constraints. The primary use of cost-matching meta-constraints is to provide an additional incentive for all constraints to be satisfied to the same degree. This type of behavior usually falls out of the normalization procedure and cost functions naturally but can depend on the distributions of the variables to be matched and the particular set of constraints that must be optimized.

One situation in which a cost-matching meta-constraint may be beneficial is to ensure that a group of soft distance

constraints that all maximize differences end up maximizing these differences to the same relative degree. One case in which this does not occur naturally is when the maximal differences that can be achieved for one constraint are possible only by minimizing the differences on another constraint. This is a frequently encountered issue in factorial designs that cross correlated variables, such as when trying to identify items for use in a 2 × 2 factorial design that crosses effects of length with effects of frequency. In this case, the correlation between the two variables reduces the extent to which each individual constraint can be satisfied. Instead, maximizing one constraint will tend to also lead to the minimization of the other (e.g., when attempting to select long/high-frequency words, increasing the values on one variable tends to reduce the values on the other). Because costs are typically squared and increasingly large costs result for the same absolute increase in the difference between two conditions, the optimal (lowest cost) set of items may, in fact, be a set that maximizes one difference while actually minimizing the other (despite both constraints stipulating that both differences should be maximized).

This problem can be rectified, however, by stipulating that the two constraints must both be minimized to the same degree. Expressed as a general cost function that minimizes the differences between the costs of constraints O_1 and O_2, this meta-constraint can be written as the difference between the current cost associated with the two constraints with the addition of the standard weight and exponents:

$$O_{MATCH}(O_1, O_2, b, n, d) = 100b(|dO_2 - O_1|)^n \quad (8)$$

An additional scaling parameter, d (default 1.0), allows for O_1 to be matched to a multiple of O_2 (e.g., so that O_1 should be satisfied only half as well as O_2, in a case where it is more important to ensure that O_2 is well satisfied). Furthermore, the equation also contains a constant multiplier of 100 because, without such a multiplier, the values of the meta-constraint tend to be on the same order of magnitude as the simple constraints over which they operate. Given that meta-constraints are typically applied when the simple constraints themselves are insufficient to adequately characterize the desired optimization, we have found that a stronger pressure to satisfy the meta-constraints is required and that the multiplier of 100 is sufficiently large to robustly provide such a pressure, without overly dominating overall cost as the optimization progresses. Notwithstanding these details, this new cost function bears a strong similarity to the cost function for minimizing differences between two measures in Eq. 2.

With the addition of this constraint, having a very small cost for one constraint and a very large cost for another will be penalized severely, whereas satisfying both constraints to the same extent results in no penalty. This shifts the lowest achievable cost value back toward the state where both constraints are satisfied to an equal degree. The degree of importance placed on ensuring that both of these constraints are equally satisfied can be emphasized by increasing the weight and/or exponent of the equation, as usual. However, the default parameter values typically produce quite a strong pressure to avoid egregious differences between two costs.

A second case that necessitates the use of a cost-matching meta-constraint involves maximizing differences between more than two conditions that span a range of values in a multilevel design—for example, when creating low-, medium-, and high-frequency conditions that are equally spaced across a range of word frequencies. Without a meta-constraint, the lowest possible cost is achieved when two of the conditions have the same mean frequency and the third is maximally different.

To illustrate this concretely, imagine that word frequencies are bounded between 1 and 11 and that the low- and high-frequency conditions have means equal to these lower and upper bounds, respectively. Without a meta-constraint, the putative medium-frequency condition would actually lower cost the most by also having a mean of 1 or 11, rather than a mean of 6 (we assume a mean of 1 in what follows). This is because the decrease in the cost associated with being very far from one condition [medium/high cost: $-(10^2) = -100$] combined with the cost of being very close to another condition [low/medium cost: $-(0^2) = 0$] results in a lower overall cost (−100) than if the medium condition's mean were located at the midpoint (6) between the low and high distributions [$-2(5^2) = -50$]. A cost-matching constraint can overcome this problem by requiring that the cost associated with the difference between the low- and medium-frequency conditions be equal to that between the medium- and high-frequency conditions. The meta-constraint will be minimized when the three conditions are equally spaced $\{2[(-25)\text{-}(-25)]^2 = 0$; yielding a total cost of −50} and maximized with two conditions of equal means and a third condition separated from the first two [$(-100 - 0)^2 = 10000$; yielding a total cost of 9,900]. An example of the successful application of this type of constraint structure is described in the examples section of the main text.

Note that in the previous discussion, we have focused on instances of matching the degree to which constraints that *maximize* differences are satisfied. This was intentional because, in the case of minimizing differences, the desired behavior is already largely accomplished by the cost-matching constraint: As the difference between the measures being compared decreases, there is a diminishing return on the overall decrease in cost as cost asymptotes toward a lower bound of zero. This behavior can nevertheless be enforced more strongly via a cost-matching constraint, but it is more likely that these types of constraints will be more useful in the former context. In both cases, however, the two constraints

being matched are both either difference minimization or difference maximization constraints; these two types generally should not be mixed. Instead, a conditional match meta-constraint is usually more suitable in those instances.

Conditional cost-matching meta-constraints Conditional cost-matching meta-constraints are a variant of the basic cost-matching meta-constraints that seek only to match the costs of two constraints when one of the constraints has not been satisfied. The primary use for these constraints is to allow the maximization of differences between variables to occur only if all of the differences to be minimized have been eliminated. This is the case when it is critical that there be virtually no differences on some measurement but there is still a desire to maximize the differences on another constraint. As for the standard cost-matching meta-constraint, this type of constraint satisfaction often occurs naturally but may require an additional pressure in some circumstances—for example, when relatively small increases on the differences to be minimized can lead to very large differences on the differences to be maximized. A pressure that prevents such behavior could be induced by adjusting the weights or exponents of the different cost functions, but these adjustments are likely to be problem specific. Conditional cost-matching meta-constraints are a more general solution to this problem.

A concrete example of a situation that might benefit from a conditional cost-matching constraint would be in the selection of stimuli for an experiment on the effects of word imageability (i.e., how easy it is to picture the meaning of the word in the "mind's eye"; Paivio, Yuille, & Madigan, 1968), using a low- and a high-imageability condition. Generally, effects of imageability are fairly weak relative to other effects, such as word frequency, so it would be desirable to identify items that are maximally different in terms of imageability, provided that there is no difference in frequency. A cost function that allows only differences for a constraint to be maximized, O_{MAX}, conditional on the differences for another constraint being successfully minimized, O_{MIN}, can be written as a function of the costs associated with these constraints and of the standard weight, exponent, and scaling parameters:

$$O_{condMATCH}(O_{MIN}, O_{MAX}, b, n, d) = \begin{cases} 100b\left(\sqrt{|dO_{MAX} - O_{MIN}|O_{MIN}}\right)^n, & \text{if } O_{MAX} < 0 \\ 0, & \text{if } O_{MAX} >= 0 \end{cases} \quad (9)$$

The square-root component of the equation serves to keep the conditional matching function on roughly the same scale as the standard cost-matching function (see the cost-matching meta-constraint description for details on the other parameters). As long as differences on the measurement to be minimized exist, O_{MIN} will be greater than zero, and there will be a pressure to reduce the differences that a simple constraint attempts to maximize. As O_{MIN} is reduced, this pressure is gradually removed, and O_{MAX} becomes a free parameter that the simple constraint will lower as much as is possible. O_{MIN} thus approximates a conditional gating function. The upper bound of 0 when O_{MAX} is greater than 0 is included to prevent undesirable behavior that is possible under this condition. Specifically, when O_{MIN} is greater than 0, without this bound, the desire to minimize the $|dO_{MAX} - O_{MIN}|$ component of the equation would effectively pressure O_{MIN} to maximize the distance between the two conditions in the direction opposite to that intended, until O_{MAX} was as large as O_{MIN}.

Hard constraints

Hard constraints are a special type of constraint that evaluate, in a true/false fashion, whether an item's value on a particular dimension satisfies the constraint. These constraints take precedence over the soft constraints and serve to exclude items that do not meet their requirements from the samples used to fill a condition. Hard constraints can be used if these constraints are known and are of a fixed value for the different variants of an optimization that a user might intend to run.

In the case that a user forces items to be initially placed in a sample that violate these constraints (e.g., by reading in a list of existing items for one condition), hard constraints will calculate a cost penalty such that every time the constraint is violated, a penalty of 1 is added. The total cost associated with hard constraints is accumulated in a special "total hard cost" pool, and the algorithm always attempts to reduce "hard costs" prior to reducing "soft costs." This allows for hard constraints to be optimized first.

Hard-bound constraints The hard bound constraint serves to constrain the value of a particular variable in a condition to fall within an upper or lower bound. This type of constraint can be useful in making minor adjustments to the items that can be included in a condition after examining the performance of an initial optimization without needing to filter out data outside of the software. Hard-bound constraints can also be used to speed up the optimization when drawing items for multiple conditions from the same

population and trying to cover different ranges of values of a variable. For instance, a low- and high-frequency condition might both sample from the same pool of words. In this case, the exact lower and upper bounds for these conditions may not be known in advance and should, therefore, be left for the optimizer to determine on the basis of how frequency differences can be maximized without violating other constraints. Rather than splitting a population into separate high- and low-frequency subpopulations, a hard constraint could be imposed to help reach this goal by setting some conservative bounds on the two conditions. This would limit the attempted swaps to not include items that would clearly never be part of the "optimal" sample without interfering with the discovery of the optimal separation point between the two groups.

Appendix 2

Tutorial and details for the ME95 SOS optimization

Here, we describe in detail the process of running an optimization in SOS. All optimizations in SOS can be accomplished simply by pointing-and-clicking within the graphical user interface (GUI), which is launched by default in the standalone versions of the software or by the sos_gui() command in MATLAB (see Fig. 3). However, users may execute the same commands by writing a simple script file in a text editor, and our example focuses on these text-based interactions with SOS. Creating these scripts is facilitated by having the SOS GUI display the script commands associated with GUI-triggered events. Scripts can subsequently be loaded and run from the GUI or, for users wanting to take advantage of SOS's advanced functionality, within MATLAB.

As our detailed example, we will examine the use of SOS to create a set of stimuli superior to those discovered in ME95, Experiment 2. This experiment studied the effects of word frequency in the absence of several confounding variables and involved 24 pairs of high- and low-frequency words that differed on frequency while being matched on AoA, length (in letters), and imageability. For use with SOS, we recreated the populations from which ME95 selected their stimuli. Words with Kučera and Francis (1967) frequencies greater than 110 per million made up the high-frequency population, and words with frequencies less than 10 per million made up the low-frequency population.

First, we will go over the commands that set up and run a greedy optimization in SOS. Then we will show how to calibrate and run a stochastic optimization. In this particular case—and in many cases researchers may encounter—the greedy version of the algorithm performs very well and can, in fact, solve the target optimization problem to a greater extent than can the stimuli reported in ME95. Only in more complex cases or when an extremely optimized set is desired may a stochastic optimization be required. However, to demonstrate the functionality of SOS and the way in which to set up a stochastic optimization, we have nevertheless included it

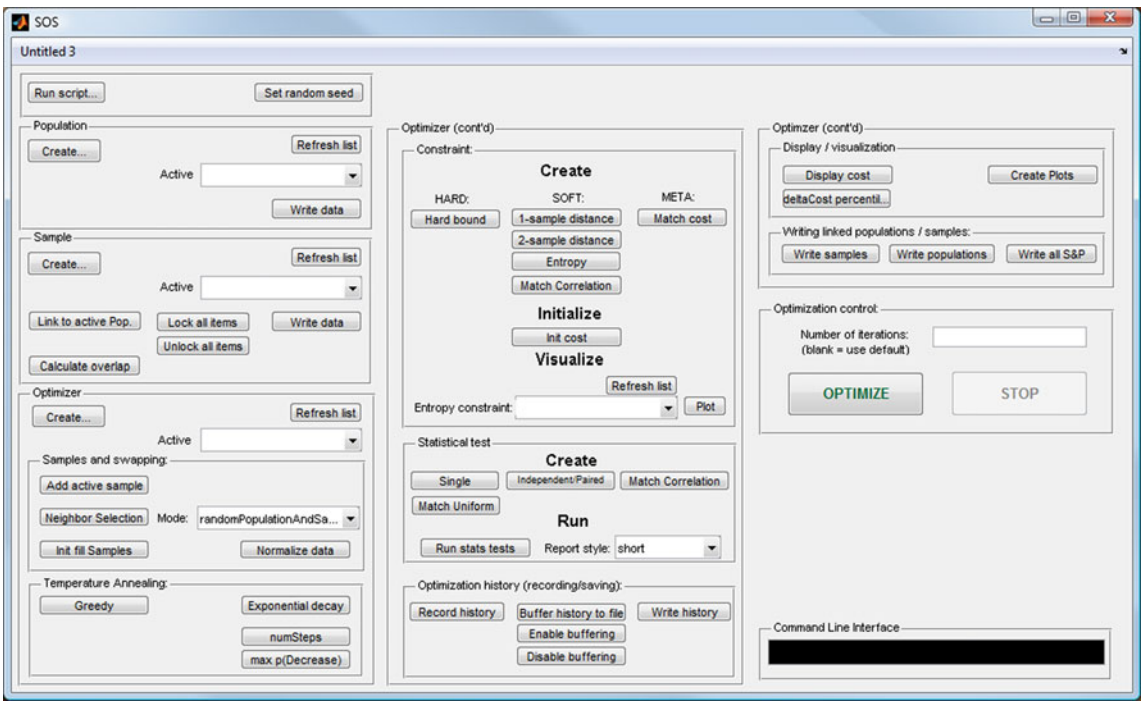

Fig. 3 The main window of the SOS graphical user interface. The process of running an optimization generally flows from top to bottom, left to right

as part of this example. Furthermore, to demonstrate how to calibrate a stochastic optimization from information garnered from an initial run of a greedy optimization, we have imposed strict limits on the greedy optimization, so that it artificially ends prematurely.

The script file for this example and all of the realistic examples reported in the main text are available through the online user manual and may be useful as templates for similar optimization problems.

Creating the samples and population Before starting to work in the SOS software proper, the files that contain the population data need to be formatted in a way that SOS recognizes. In this case, we have two such files: a high-frequency population and a low-frequency population. The items need to be in tab-delimited columns, with one row corresponding to one item. Data sets containing missing values are currently not supported. Each column should also include a label, or header, at the top of each column so that it can be referred to easily later (although SOS will automatically label the columns if headers are not specified). Additionally, although SOS attempts to auto-detect the type of information present in each column (i.e., words or numbers), users can include this formatting information explicitly by placing a vertical bar ("|") after the column name and either an "s" indicating word/nonnumeric information (strings) or an "f" indicating that the column contains (floating-point) numbers. For example, the first three columns and two rows of our high-frequency population file appear as follows:

```
word|s   AoA|f   KFfrequency|f

act      3.19    237
```

Once the data files have been formatted, the SOS software can be launched, and the data can be read in and used to create populations and samples. To do this, we first create the populations from which the optimized stimuli will be drawn. In this case, we are using as our populations two text files that contain high- and low-frequency words according to the constraints imposed by ME95 and that contain header and formatting information.

```
highFreqPopulation = population('Exp2HighFreqPopulation.txt', 'name', ...
                'highFreqPopulation', 'isHeader', true, 'isFormatting', true);


lowFreqPopulation = population('Exp2LowFreqPopulation.txt', 'name', ...
                'lowFreqPopulation', 'isHeader', true, 'isFormatting', true);
```

The **population** command creates a new population called "'highFreqPopulation'" from our text file, "'Exp2HighFreqPopulations.txt'." Because the columns in this file are labeled with headers and formatting information, we have set the "'isHeader'" and "'isFormatting'" parameters to "true." In a different scenario, a researcher may want all lists of words to draw from the same population. In fact, in the present example, we could have used a single population of words and then, by using hard-bound constraints, imposed the same frequency cutoffs imposed by ME95 (see Appendix 1). However, we opted to keep our procedure as similar to that in ME95 as possible, and so, we created two separate populations of words. Additional examples in the main text and user manual describe successful optimizations with this alternative constraint and population structure.

After these populations have been created, we can create the samples that we want to optimize.

```
highFreqSample = sample(24, 'name', 'highFreqSample', ...
                'outFile', 'highFreqSampleGREEDY.txt');


highFreqSample.setPop(highFreqPopulation);
```

Above, a sample of 24 words called "'highFreqSample'" is created using the **sample** command. When the optimization ends, this sample can be saved to a file called "'highFreqSample.txt'." We then link this sample with the population from which its words will be drawn ("'highFreqPopulation'"). Similar commands are repeated to prepare a low-frequency sample, "'lowFreqSample'."

Creating the optimization At this point, we can create the optimization itself.

```
Exp2GreedySOS = sos('reportInterval', 100, 'stopFreezeIt', 100, ...
                    'statInterval', 500, 'blockSize', 100);
```

With the **sos** command, a new instance of an SOS optimization with the name "'Exp2GreedySOS'" is created. There are several advanced options that may also be set with this command (e.g., "'reportInterval'," "'stopFreezeIt'," "'statInterval'"; see the user manual for details); if not set, however, these options take on their default values. In this case, we have instructed SOS to provide updates and statistical tests more frequently during the optimization process than the default values. Additionally, to later demonstrate the configuration of a stochastic optimization, we have set the number of iterations after which the algorithm will end if cost remains unchanged ("'stopFreezeIt'") to be artificially low and changed the block size over which $\Delta cost$ will be recorded when calculating the distribution of cost changes to the same value. As a result, the greedy optimization will end early but will still generate a $\Delta cost$ distribution corresponding to that expected in a minimum—that is, only positive $\Delta cost$ values—during the last block of iterations.

Once the optimization has been created, we explicitly link the samples to be optimized (i.e., "'highFreqSample'" and "'lowFreqSample'") to the optimization (i.e., "'Exp2GreedySOS'") using the **addSample** command—for instance, to add the high-frequency sample,

```
Exp2GreedySOS.addSample(highFreqSample);
```

Adding the constraints Next, we provide the optimizer with knowledge of the properties we want the final samples to have; in this case, we want the samples to differ on frequency and be matched on AoA, length, and imageability. To give the optimizer this knowledge, each of these properties is translated into a constraint and an associated cost function for operationalizing this constraint. Using the **addConstraint** command, we can add to the optimization the first constraint: to maximize the differences between the lists on frequency.

```
freqConstraint = Exp2GreedySOS.addConstraint( ...
                    'name', 'freqConstraint', 'constraintType', 'soft', ...
                    'fnc', 'orderedMax', 'stat', 'mean', ...
                    'sample1', lowFreqSample, 'sample2', highFreqSample, ...
                    's1ColName', 'KFfrequency', ...
                    'S2ColName', 'KFfrequency', ...
                    'paired', true);
```

The type of constraint we are using here is "'soft'" (see the user manual for other options). The "'fnc'" parameter is set to "'orderedMax'" to instruct the optimization to maximize the differences between the samples on the target statistic, which has been specified as the "'mean'" via the "'stat'" parameter. In the present context, this constraint ensures that "'sample1'" ("'lowFreqSample'") will have lower average frequency than "'sample2'" ("'highFreqSample'").

Next, we specify the names of the samples and the names of the variables (columns) within each sample to which this constraint pertains. Because we want to make sure that each *pair* of words differs as greatly as possible on frequency (and because ME95 adopted a pairwise selection procedure), "'paired'" is set to "true." Advanced parameters (e.g., "'weight'" and "'exponent'") may also be configured using the **addConstraint** command (see the user manual for details). If left unspecified, as in the present example, they will be set to their default values.

Next, we use the same command to create the second constraint: matching the highFreqSample and lowFreqSample on AoA. This involves using the same command to add a new constraint but setting the "'fnc'" parameter to "'min'," as well as changing the variables over which this constraint should operate.

```
freqConstraint = Exp2GreedySOS.addConstraint( ...
                    'name', 'freqConstraint', 'constraintType', 'soft', ...
                    'fnc', 'orderedMax', 'stat', 'mean', ...
                    'sample1', lowFreqSample, 'sample2', highFreqSample, ...
                    's1ColName', 'KFfrequency', ...
                    'S2ColName', 'KFfrequency', ...
                    'paired', true);
```

The remaining two constraints, minimizing the pairwise differences between the two lists on length and imageability, follow this same format.

Finally, we have also added three meta-constraints to constrain the costs associated with the simple constraints. In particular, these simple constraints consist of one maximization constraint (frequency) and three minimization constraints (AoA, length, and imageability). While the cost values associated with minimization constraints have a lower bound of zero, a maximization constraint's cost does not have a lower bound and can become negative. By virtue of this difference between the two kinds of constraints, one or more maximization constraints can sometimes dominate the cost function and cause other constraints to be more poorly satisfied (as assessed in a pilot optimization without these meta-constraints). This type of problem can be avoided with the use of meta-constraints. One type of meta-constraint, "'matchCostNotMin'," effectively prevents a maximization constraint from being optimized until a minimization constraint has been satisfied. In the present example, we will use three "'matchCostNotMin'" meta-constraints, one to pair each minimization constraint with the "'orderedMax'" frequency constraint.

```
metaAoAAndFreqConstraint = Exp2Greedy SOS.addConstraint(...
                    'name', 'metaAoAAndFreqConstraint', ...
                    'constraintType', 'meta', ...
                    'fnc', 'matchCostNotMin', ...
                    'constraint1', AoAConstraint, ...
                    'constraint2', frequencyConstraint);
```

To create a "'matchCostNotMin'" meta-constraint, we change "'constraintType'" to "'meta'" and "'fnc'" to "'matchCostNotMin'." Then, we enter the names of the two constraints to be matched. Importantly, when using "'matchCostNotMin'" constraints, the name of the minimization constraint must occur as 'constraint1'. The format for creating the remaining

two meta-constraints that equate the frequency constraint to the length and imageability constraints is the same as above.

Now that we have created the optimization and the constraints, we can prepare to run the optimization by initially filling the two sample lists. With the **initFillSamples** command, items are selected randomly from the populations to fill our high- and low-frequency samples.

```
Exp2GreedySOS.initFillSamples();
```

Statistical tests Optionally, a researcher may define statistical criteria for the optimization. If these criteria are met, the optimization will end. Because statistically significant differences or, alternately, nonsignificant "matches" between samples are often all a researcher wants to achieve (and are the kind of evidence presented to demonstrate that the stimuli were well suited for studying the target question when results are published), further optimization may be superfluous once reasonable statistical criteria have been met. Furthermore, as is discussed in more detail in the user manual, additional processing to the point of reaching a cost minimum may constrain the number of sets of items that can satisfy the constraints, thereby interfering with the generalizability of the results.

In the present example, each constraint is associated with a single statistical test. In the case of word frequency, we would like to ensure that the means of the two samples are statistically significantly different, as follows.

```
Exp2GreedySOS.addttest( 'name', 'freqTest', ...
                        'type', 'paired',...
                        'sample1', highFreqSample, 'sample2', lowFreqSample, ...
                        's1ColName', 'KFfrequency', 's2ColName', 'KFfrequency', ...
                        'desiredpvalCondition', '<=',...
                        'desiredpval', 0.05);
```

Using the **addttest** command, we can associate this statistical test with the current SOS optimization. Because we are matching or maximizing the differences between the two lists on a pairwise basis, the type of *t*-test we will use is a paired samples *t*-test between the "'KFfrequency'" columns in "'highFreqSample'" and "'lowFreqSample'." We want these two samples to be significantly different on word frequency, so the *p*-value returned by the *t*-test should be less than or equal to .05 to stop the optimization. The parameters "'desiredpvalCondition'" and "'desiredpval'" together specify this statistical criterion.

In the case of the AoA constraint (as well as the length and imageability constraints), a different statistical criterion is required to infer that the samples are highly similar to one another, and, unlike statistical significance for rejecting the null hypothesis, there is no gold standard *p*-value for means that do *not* differ. For this example, we assume that the two lists are adequately matched if the *p*-value returned by a paired samples *t*-test is greater than or equal to .5. This value of .5 is somewhat arbitrary, and other researchers may wish to match to a greater (e.g., *p*-value of .9) or lesser (e.g., *p*-value of .3) degree, depending upon the problem at hand and existing standards in their fields. By comparison, in Experiment 2, ME95 were able to achieve "matches" of $p = 1.0$ for word length, $p = .20$ for AoA, and $p = .01$ (i.e., a failed match) for imageability, using manual stimulus selection.

Configuring and running the optimization In the last portion of this example script, we will explain how to run both greedy and stochastic optimizations; the steps for both up to this point are the same. As a first step for all researchers, we recommend running a greedy optimization. Many problems are solved with this type of optimization, and it generally takes less time than a stochastic one. Additionally, stochastic optimizations are calibrated with information calculated during a run of the optimization in greedy mode, so a user does not lose any time from beginning a problem in this way. In the case of our ME95 example, we show that, although the greedy optimization finds sufficiently optimal high- and low-frequency sets of words, the stochastic version of SOS performs even better.

To visualize the progress of the optimization, we have included an optional command to create the graphs associated

with various critical measures during the optimization, such as cost and the *p*-values associated with each statistical test.

```
Exp2GreedySOS.createPlots();
```

Greedy optimization The optimizer defaults to greedy optimization; that is, it always swaps to items that result in lower overall cost, and the optimization never behaves stochastically. We are therefore ready to run the greedy optimization by clicking the 'Optimize' button in the GUI (or typing Exp2GreedySOS.optimize() at the command prompt, although the command prompt interface does not currently allow for ad hoc stopping and starting of the optimization by the user). At the beginning of an optimization, SOS calculates the initial value of cost—the cost associated with the items that have been randomly selected to fill our high- and low-frequency samples—and displays the results.

```
Initializing Constraints

    Hard Constraint Total:  0

    Soft Constraint #:1 Cost:  -1.6583 freqConstraint

    Soft Constraint #:2 Cost:  0.62044 AoAConstraint

    Soft Constraint #:3 Cost:  1.1733 lengthConstraint

    Soft Constraint #:4 Cost:  1.01 imageConstraint

Soft Constraint Total:  1.1455

    Meta Constraint #:1 Cost:  141.3792 metaAoAAndFreqConstraint

    Meta Constraint #:2 Cost:  332.2222 metaLengthAndFreqConstraint

    Meta Constraint #:3 Cost:  269.5026 metaImagAndFreqConstraint

Meta Constraint Total:  743.104


TOTAL COST (soft + meta):  744.2495
```

This display reveals that the three minimization constraints are associated with roughly equal costs before the optimization begins. Because of the cutoffs we imposed on the high- and low-frequency lists outside of the optimization, the frequency maximization constraint is already associated with a negative cost value, indicating that the low-frequency list already has lower pairwise frequency than the high-frequency list. Finally, the values of the meta-constraints are by design approximately two orders of magnitude larger than those of the soft constraints; in practice, we have found that this implements a reasonably strong but not overwhelming pressure for these constraints to be well satisfied.

Next, the SOS software begins to print out the values of cost after a specified number of iterations have passed; in this case, we set the algorithm to report the cost value every 100 iterations. Along with this cost information, the estimated time remaining in the optimization (assuming that the maximum number of iterations—in this case, 10,000—has been reached) and the percent of the optimization completed are also displayed. Periodically, SOS assesses whether the user-specified statistical tests have passed their criteria.

```
Optimizing for 10000 iterations
 Iteration  Cost       % Complete  Elapsed  Remaining
 1)         744.2495   0.01%       1s       1m 10s
 100)       144.85112  1.00%       1s       2m
 200)       76.94003   2.00%       2s       1m 31s
 300)       52.11457   3.00%       2s       1m 17s
 400)       37.08432   4.00%       3s       1m 9s
 500)       32.87713   5.00%       3s       1m 4s

Running all stat tests:
UserHyp:  PASS; highFreqSampleKFfrequency - lowFreqSampleKFfrequency:
t[paired](23) = 15.5669, p = 1.049e-13 p-des:  0.05 m(1) = 162.7083;
m(2) = 4.2083 (se=2.0784)
UserHyp:  FAIL; highFreqSampleAoA - lowFreqSampleAoA: t[paired](23) =
-0.40505, p = 0.68918 p-des:  0.5 m(1) = 3.8988; m(2) = 3.9358
(se=0.018688)
UserHyp:  PASS; highFreqSampleletters - lowFreqSampleletters:
t[paired](23) = 1.4968, p = 0.14803 p-des:  0.5 m(1) = 5.875; m(2) =
5.5833 (se=0.039774)
UserHyp:  FAIL; highFreqSampleimagery - lowFreqSampleimagery:
t[paired](23) = -1.1709, p = 0.25362 p-des:  0.5 m(1) = 4.7254; m(2) =
4.8396 (se=0.019902)
```

Over the first 500 iterations of the optimization, cost has decreased rapidly. The frequency difference between the two samples is statistically significant, and the two samples are matched pairwise on length to a greater degree, in fact, than we had required. However, the statistical tests reveal that the high- and low-frequency samples are still relatively different on AoA and imageability. Here, an example of a "full" statistical report is shown, but a "short" report is actually the default. The choice of report style may be selected from the GUI or with an optional argument when creating the SOS object (see the online manual for details).

This greedy optimization continues for only 81 more iterations. At this point, the optimization ends because cost has "frozen" and remained unchanged for our prespecified number of iterations (100). This number was selected to artificially stop the greedy version, to allow us to also demonstrate the method used to configure stochastic optimization; increasing the number of iterations during which cost remains unchanged

before stopping provides an increasingly confident measure that cost has descended into a (potentially local) minimum. In addition to stopping due to frozen cost, other possible termination messages include "all statistical test passed defined criteria" or "the [user-specified] maximum number of iterations has been reached [default of 10000]."

When this optimization ends, the high-frequency sample has a significantly higher frequency than does the low frequency sample, paired $t(23) = 15.57$, $p = 1.01 \times 10^{-13}$. On the other hand, the two lists do not differ significantly on AoA, paired $t(23) = -0.41$, $p = .69$, word length, paired $t(23) = 1.50$, $p = .15$, or imageability, paired $t(23) = -1.17$, $p = .25$. Using even the intentionally limited greedy version of SOS with a small freezing interval resulted in arguably better matches than those selected by ME95 by hand—and, we assume, in far less time.

Additional insight into the progress of the optimization can be obtained by examining the plots produced during the optimization. The plots associated with the current optimization are shown in Fig. 4. The top plot displays the value of temperature throughout the optimization. Because greedy mode is equivalent to setting temperature to zero for the entire optimization, the temperature plot shows no fluctuations. The next plot displays the value of cost and shows a sharp decrease at the beginning of the optimization, followed by very gradual progress toward the final low cost value. The third plot displays the $\Delta cost$ value associated with a swap at each iteration for which data are being plotted and gives a sense of the variability and sign of the cost differences encountered after an attempted swap. For example, large $\Delta cost$ variability indicates high variability in the optimality of the current set, relative to the swap set, and a large number of positive $\Delta cost$ values could indicate that the optimization is becoming stuck in a minimum. The fourth plot shows the probability, p(swap), of moving to a neighboring (swap) item, averaged across all of the iterations in a block (by default, of size 1,000). Low p-values on this plot indicate that the algorithm is not moving to neighboring states and provides additional evidence that it could be becoming trapped in a minimum. High p(swap) values indicate that the algorithm is swapping frequently between states, a scenario encountered during the initial stages of the stochastic version of the algorithm or when most items in the population could be selected to satisfy the constraint to the same degree. The last plot shows the p-values associated with the four statistical criteria defined for this example on the basis of the order in which they were created. The 'Names' button shown on the left of the figure displays the full name of the statistical tests in an overlay, not shown. In our present example, the p-value for Test #1 (frequency) remains low throughout the optimization. The other three tests, associated with the minimization constraints, gradually move to the final state where all p-values are greater than .05. We can see

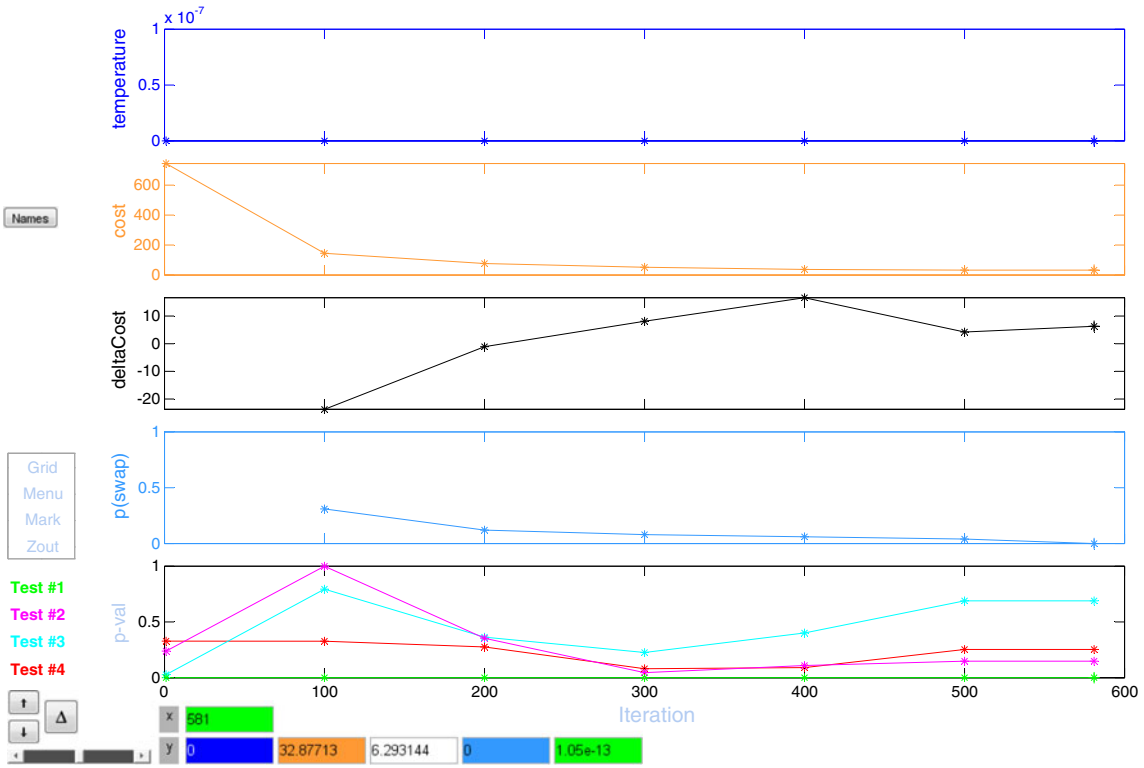

Fig. 4 Plot generated by the SOS software of values for several metrics throughout the course of the greedy optimization. The topmost plot displays temperature, the second plot displays cost, the third plot displays $\Delta cost$, the fourth plot displays the likelihood of swapping to a neighboring item, p(swap), and the last plot displays the p-values for the statistical tests. See the text for details

that Test #2 (AoA) remained high throughout the optimization, while more time was required to minimize the constraints associated with Tests #3 (length) and #4 (imageability).

Stochastic optimization Next, we demonstrate the use of the stochastic version of SOS. To do so, we have used the same example case from ME95 and the same basic script and initial sample of stimuli. In fact, to ensure that greedy and stochastic runs are initialized identically, we used the **setSeed** function to initialize each with the same random number (not shown in the first example), meaning that the high- and low-frequency samples will be filled with the same initial items. Within the script, however, we have changed the annealing schedule to use exponentially decaying temperature annealing instead of the default greedy mode (see the user manual for details). Before reviewing how this is accomplished, however, it is necessary to first collect some information to allow for the calculation of *pDecrease*, a parameter in the exponential decay function that specifies the next temperature level as a smaller proportion of the current temperature. The goal here is to identify a value of pDecrease that gradually moves from a high temperature, at which swaps occur at random, to a lower temperature that would allow for a small number of swaps to still occur when the algorithm enters a cost minimum. This will allow the algorithm to gradually decrease cost and avoid becoming stuck in a local minimum as it does so.

To determine the appropriate value of pDecrease for an optimization, we offer the following strategy. All optimizations should begin by attempting success through a greedy run of SOS. If a satisfactory solution is not found, users should then extract some additional information from the greedy optimization before beginning a stochastic one. Specifically, in the previous optimization, the command **Exp2GreedySOS.deltaCostDeciles** will produce the distribution of Δ*cost* values from the last iteration block in the form of a cumulative average (block size is 1,000 by default, but we have changed it here to match our artificially low "'stopFreezeIt'" of 100). In the present example, the delta cost deciles are as follows:

```
Deciles for deltaCost in last block:
(100 iterations total; numIt < blockSize in first block)
0:  +0.61125129
10:  +3.87051217
20:  +6.11557827
30:  +6.95036521
40:  +8.88808587
50:  +10.09334703
60:  +11.16903585
70:  +13.83542952
80:  +15.75305910
90:  +19.65537253
100:  +32.39223807
97.5th - 2.5th percentile deltaCost:  29.10367106
```

Now that we know the distribution of changes in cost during the final stages of a greedy optimization, when it potentially becomes stuck in a minimum in which further decreases are not possible, we can infer a "final" temperature value that will still allow for some cost-increasing swaps to occur for some percentage of iterations when the optimization approaches this minimum. To determine the initial value of temperature, we next run the optimization in stochastic mode for one block of iterations; during this time, swaps occur at random. After this block completes, we again display the Δ*cost* percentiles (not shown). Armed with this information from greedy and stochastic optimizations, we are able to configure the exponentially decaying temperature function. We have found that setting the upper (initial) temperature of the optimization to the 97.5th − 2.5th percentile of Δ*cost* values from the first block of stochastic trials calibrates the algorithm's initial temperature to a reasonable initial value. Specifically, this value is sufficiently high for most swaps to be made effectively at random but is relatively stable across different optimizations that are initialized with different random seeds (in those instances, the increased variability of using the full range of Δ*cost* values leads to substantial variability in the estimates of pDecrease). We also recommend initially using the 10th percentile of Δ*cost* values from the greedy optimization as the lower (final) temperature. Choosing a smaller or larger percentile for final Δ*cost* will lead to faster or slower annealing, respectively, with faster annealing settling more rapidly into a minimum, albeit with a greater risk that this minimum is local and not global. Finally, the calculation calls for the number of temperature steps (decreases) that the algorithm will take as it goes from the initial temperature to the final temperature; we recommend at least 10 (and never less than 3, since fewer steps are effectively no different than starting a greedy search with different initial sets of items). Below, we show the command that calculates pDecrease and its output:

```
expAnneal.maxpDecrease(667.49, 3.87, 10)
```

```
0.402508 is max pDecrease to ensure 10 steps during exp anneal
```

Subsequently, we use the following command to indicate that we want to run a stochastic optimization and to set pDecrease to the value we have calculated:

```
Exp2StochasticSOS.setAnnealSchedule('schedule', 'exp', 'pDecrease', .402508);
```

This changes the optimization from "'greedy'" to "'exp'" (short for "exponentially decaying temperature annealing"). With this same command, we have also set an additional parameter that governs the rate at which temperature decreases during the optimization, "'pDecrease'," to the value we have just calculated.

Now that we have calibrated the stochastic optimization, we are ready to run it. As in the previous greedy example, the optimization begins by calculating the initial value of cost; because the high- and low-frequency samples are filled with the same items in both, the initial cost values will be identical between greedy and stochastic runs. Below are displayed the first 3,000 iterations of the optimization. Note that the maximum number of iterations has been changed here to be larger than the default value to allow the optimization to run for more iterations, as is generally required in stochastic optimizations.

```
Optimizing for 1000000 iterations
 Iteration  Cost         % Complete  Elapsed  Remaining
 1)         744.24945    0.00%       1s       1h 57m 50s
 1000)      3149.89991   0.10%       7s       1h 54m 3s

1000) Annealing Equation calibrated, changing temperature from Inf to
667.4918
 2000)  1753.88764  0.20%  13s  1h 47m 11s

2000) p(thermEquil):  1.6257e-10 prevBlock m = 1871.7707 (se = 0.79726)
curBlock m = 1679.0243 (se = 0.51208)
 3000)  1368.77390  0.30%  19s  1h 44m 28s

3000) p(thermEquil):  2.8947e-92 prevBlock m = 1679.0243 (se = 0.51208)
curBlock m = 1244.0813 (se = 0.37948)
```

Following the initial block of iterations in which temperature is effectively set to an infinitely high value, the display shows the completion of the calibration of the initial temperature value, and the algorithm will gradually lower temperature throughout the course of the optimization. A simple way of doing so would be to simply lower temperature after a fixed number of iterations have passed. However, we have found that doing so is often suboptimal because the algorithm will sometimes spend either too much time at high temperatures, where swaps are mostly random, or too little time at an optimal low temperature that allows for local minima to be avoided reliably. Instead, the algorithm employs a more sophisticated procedure for evaluating when temperature should be lowered, referred to as an "assessment of thermal equilibrium." A full discussion of this algorithm is not necessary at present and is beyond the scope of the present article (see the user manual for details). In essence, however, this procedure lowers temperature only once it appears likely that such a lowering of temperature will not result in the algorithm's becoming immediately stuck in a local minimum. This can be approximated by evaluating when cost does not change significantly across subsequent blocks of trials.

Each assessment of thermal equilibrium lists the means and standard errors of the cost values encountered in the previous and the current blocks of trials and the probability that these values are not equal. In our example, thermal equilibrium for this T_0 is not reached until 7,000 iterations.

```
7000) p(thermEquil):  0.75543 prevBlock m = 1431.404 (se = 0.33026)
curBlock m = 1436.5575 (se = 0.40568)
7000) Thermal Equilibrium Reached - Dropping temperature from 667.4918
to 398.821
```

At this point, temperature is reduced according to the value of pDecrease; in this case, temperature is decreased by approximately 40% whenever thermal equilibrium is reached. Figure 5 displays the plots generated during the stochastic version of this optimization and reveals additional information about the procedure. During the early iterations, temperature is set to a high value, and so the Δ*cost* values vary substantially both positively and negatively because the algorithm is effectively swapping to different items at random. This behavior is directly reflected in the probability of swapping to a neighboring item, *p*(swap), which is equal to .5 for these early iterations. Similarly, the statistical tests shown at the bottom of the figure vary wildly from iteration to iteration. Eventually, temperature begins to decrease, and, as a result, cost values decrease and become more stable over time. Near the end of the optimization, *p*(swap) has been reduced to near zero; at this point, the algorithm is effectively swapping only to items that strictly decrease cost. However, even at the end of the optimization, the *p*-values from the statistical tests appear to be changing; this result suggests that changing even a single item may have a nontrivial impact on the statistics in this particular optimization. After 210,000 iterations, the algorithm stops because all of the statistical criteria have been met. At this point, high- and low-frequency samples have been selected that vary on frequency, paired $t(23) = 16.13$, $p = 4.92x10^{-14}$, but not on AoA, paired $t(23) = -0.68$, $p = .51$, length, paired $t(23) = 0.33$, $p = .75$, or imageability, paired $t(23) = 0.39$, $p = .70$. These results indicate that the stochastic optimization identified a better set of items than did both the greedy optimization and ME95.

Assessing generalizability In SOS, we can also assess the degree to which the final samples of stimuli are representative of the underlying population of stimuli. If, instead, they are an idiosyncratic subset, we cannot make strong claims about the generalizability of any experimental effects based on inferential statistics. In the case of the high- and low-frequency word lists, we ran five separate stochastic optimizations until each passed the statistical criteria, yielding five lists of high- and five lists of low-frequency items. We then used the "calculate overlap" command [of the form **dataFrame.overlap(sample1run1,sample1run2)**] to determine how similar the resulting lists in each condition were to each other. On average, the high-frequency lists shared 12.92% (*SD* = 0.09) of their items, while the low-frequency lists shared 7.5% (*SD* = 0.06) of their items. These numbers suggest that the five optimal solutions were, in fact, quite different from one another and that the algorithm was not arriving at the same, unique solution each time. Thus, we would expect any behavioral effects observed with these stimuli to generalize to new sets of stimuli, as well.

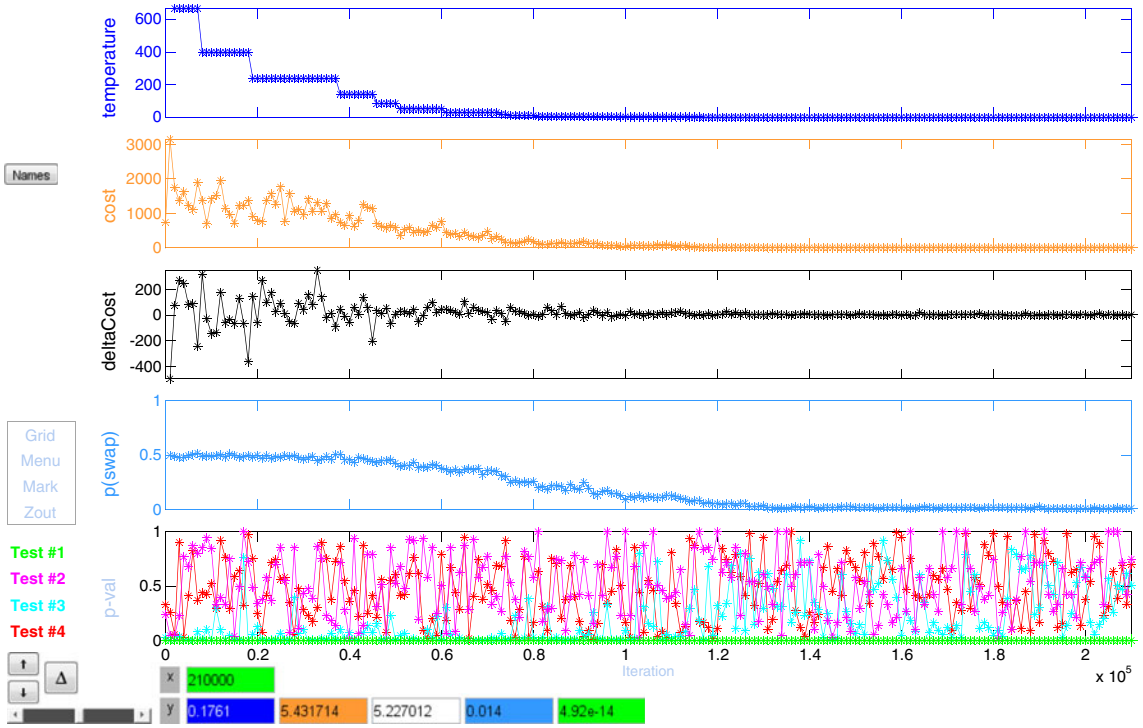

Fig. 5 Plot generated by the SOS software of values for several metrics throughout the course of the stochastic optimization. A more detailed description of the plots is included in the caption for Fig. 4 and in the text

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9,* 147–169.

Armstrong, B. C. (2007). *Comprehending ambiguous words: Computational and empirical investigations*. Masters thesis, National Archives of Canada.

Armstrong, B. C., & Plaut, D. C. (2008). Settling dynamics in distributed networks explain task differences in semantic ambiguity effects: Computational and behavioral evidence. In B. C. Love, K. McRae, & V. M. Sloutsky (Eds.), *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 273–278). Austin, TX: Cognitive Science Society.

Armstrong, B. C., & Plaut, D. C. (2011). Inducing homonymy effects via stimulus quality and (not) nonword difficulty: Implications for models of semantic ambiguity and word recognition. In T. S. L. Carlson & C. Hölscher (Eds.), *Proceedings of the 33rd Annual Conference of the Cognitive Science Society* (pp. 2223–2228). Austin, TX: Cognitive Science Society.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language, 59,* 390–412.

Balota, D., Cortese, M., Sergent-Marshall, S., Spieler, D., & Yap, M. (2004). Visual word recognition of single-syllable words. *Journal of Experimental Psychology: General, 133,* 283–316.

Beretta, A., Fiorentino, R., & Poeppel, D. (2005). The effects of homonymy and polysemy on lexical access: An MEG study. *Cognitive Brain Research, 24,* 57–65.

Binder, J., Desai, R., Graves, W., & Conant, L. (2009). Where is the semantic system? A critical review and meta-analysis of 120 functional neuroimaging studies. *Cerebral Cortex, 19,* 2767–2796.

Brainard, D. H. (1997). The Psychophysics Toolbox. *Spatial Vision, 10,* 433–436.

Brysbaert, M., & New, B. (2009). Moving beyond Kučera and Francis: A critical evaluation of current word frequency norms and the introduction of a new and improved word frequency measure for American English. *Behavior Research Methods, 41,* 977–990.

Bunn, E., Tyler, L., & Moss, H. (1998). Category-specific semantic deficits: The role of familiarity and property type reexamined. *Neuropsychology, 12,* 367–379.

Clark, H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of Verbal Learning and Verbal Behavior, 12,* 335–359.

Coltheart, M. (1981). The MRC psycholinguistic database. *Quarterly Journal of Experimental Psychology, 33,* 497–505.

Cortese, M., & Khanna, M. (2007). Age of acquisition predicts naming and lexical-decision performance above and beyond 22 other predictor variables: An analysis of 2,342 words. *Quarterly Journal of Experimental Psychology, 60,* 1072–1082.

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.

Davis, C. (2005). N-WATCH: A program for deriving neighborhood size and other psycholinguistic statistics. *Behavior Research Methods, Instruments, & Computers, 37,* 65–70.

Gernsbacher, M. (1984). Resolving 20 years of inconsistent interactions between lexical familiarity and orthography, concreteness, and polysemy. *Journal of Experimental Psychology: General, 113,* 256–281.

Gilhooly, K., & Logie, R. (1980). Age-of-acquisition, imagery, concreteness, familiarity, and ambiguity measures for 1,944 words. *Behavior Research Methods, 12,* 395–427.

Gilhooly, K., & Logie, R. (1982). Word age-of-acquisition and lexical decision making. *Acta Psychologia, 50,* 21–34.

Healy, M. J. R. (1968). Multiple regression with a singular matrix. *Journal of the Royal Statistical Society: Series C, 17,* 110–117.

Hino, Y., & Lupker, S. (1996). Effects of polysemy in lexical decision and naming: An alternative to lexical access accounts. *Journal of Experimental Psychology: Human Perception and Performance, 22,* 1331–1356.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations* (pp. 282–317). Cambridge, MA: MIT Press.

Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics, 23,* 462–466.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220,* 671–680.

Kučera, H., & Francis, W. N. (1967). *Computational analysis of present-day American English*. Providence, RI: Brown University Press.

Morrison, C., & Ellis, A. (1995). Roles of word frequency and age of acquisition in word naming and lexical decision. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 21,* 116–133.

Paivio, A., Yuille, J., & Madigan, S. (1968). Concreteness, imagery and meaningfulness values for 925 nouns. *Journal of Experimental Psychology, 76,* 1–25.

Patterson, K., & Plaut, D. (2009). Shallow draughts intoxicate the brain: Lessons from cognitive science for cognitive neuropsychology. *Topics in Cognitive Science, 1,* 39–58.

Raaijmakers, J., Schrijnemakers, J., & Gremmen, F. (1999). How to deal with "The language-as-fixed-effect fallacy": Common misconceptions and alternative solutions. *Journal of Memory and Language, 41,* 416–426.

Rodd, J., Gaskell, G., & Marslen-Wilson, W. (2002). Making sense of semantic ambiguity: Semantic competition in lexical access. *Journal of Memory and Language, 46,* 245–266.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986). Parallel distributed models of schemata and sequential thought processes. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models* (pp. 7–57). Cambridge, MA: MIT Press.

Salazar, R., Plastino, A., & Toral, R. (2000). Weakly nonextensive thermostatistics and the Ising model with long-range interactions. *European Physical Journal B, 17,* 679–688.

Schilling, H., Rayner, K., & Chumbley, J. (1998). Comparing naming, lexical decision, and eye fixation times: Word frequency effects and individual differences. *Memory & Cognition, 26,* 1270–1281.

Schneider, W., Eschman, A., & Zuccolotto, A. (2002). E-Prime, Version 1.1 [Computer Software].

Snodgrass, J. G., & Vanderwart, M. (1980). A standardized set of 260 pictures: Norms for name agreement, image agreement, familiarity, and visual complexity. *Journal of Experimental Psychology: Human Learning and Memory, 6,* 174–215.

Stanovich, K. E. (1997). *How to think straight about psychology* (5th ed.). Reading, MA: Addison Wesley.

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

Watson, C. E. (2009). *Computational and behavioral studies of normal and impaired noun/verb processing*. Unpublished doctoral dissertation, Carnegie Mellon University.

Watson, C. E., & Chatterjee, A. (2012). A bilateral frontoparietal network underlies visuospatial analogical reasoning. *NeuroImage, 59*, 2831–2838.

Woollams, A., Lambon Ralph, M., Plaut, D., & Patterson, K. (2007). SD-squared: On the association between semantic dementia and surface dyslexia. *Psychological Review, 114,* 316–339.