Behavior Research Methods
2007, 39 (4), 973-978

Match: A program to assist in matching the conditions of factorial experiments

Maarten van Casteren and Matthew H. Davis
MRC Cognition and Brain Sciences Unit, Cambridge, England

In most experiments that involve between-subjects or between-items factorial designs, the items and/or the participants in the various experimental groups differ on one or more variables, but need to be matched on all other factors that can affect the outcome measure. Matching large groups of items or participants on multiple dimensions is a difficult and time-consuming task, yet failure to match conditions will lead to suboptimal experiments. We describe a computer program, "Match," that automates this process by selecting the best-matching items from larger sets of candidate items. In most cases, the program produces near-optimal solutions in a matter of minutes and selects matches that are typically superior to those obtained using hand matching or other semi-automated processes. We report the results of a case study in which Match was used to generate matched sets of experimental items (words varying in length and frequency) for a published study on language processing. The program was able to come up with better-matching item sets than those hand-selected by the authors of the original study, and in a fraction of the time originally taken up with stimulus matching.

Factorial experimental designs typically involve contrasting two or more sets of experimental items or participants that differ on some critical dimension and yet are matched on all other dimensions that could conceivably affect the dependent measure(s) that are obtained. For example, in experiments on language processing, researchers might want to contrast response times (RTs) to two sets of words that differ on a property of interest (e.g., imageability) and yet are matched on all other properties that affect RT (e.g., frequency of occurrence, bigram frequency, number of syllables, number of letters, etc.). Similarly, in studies comparing a group of abnormal participants with a control population, researchers typically might want to select control participants that are matched to the experimental group on a number of factors that could affect performance, such as age, sex, IQ, educational level, handedness, and so on.

In both of these scenarios, scientific progress will inevitably increase the difficulty of producing matched sets of items or participants. Every study that discovers a factor that affects an experimental measure introduces another potential confound that must be controlled in future experimentation. In a psycholinguistic context, Anne Cutler noted 25 years ago that "making up materials is a confounded nuisance" (Cutler, 1981) and expressed a doubt as to whether it would even be possible to perform experimental comparisons of different language materials in the future. Nonetheless, factorial experiments remain the standard tool of experimental psychology and many other fields.

Help is at hand for the frustrated researcher. For many properties that could influence the response to specific items, databases that will allow stimulus matching already exist (e.g., the CELEX database for language research; Baayen, Piepenbrock, & Gulikers, 1995). Furthermore, whenever new properties of stimuli are discovered that affect RT in a number of tasks, the research communities will often circulate databases of measured norms of all the relevant properties. Similarly, for researchers interested in comparing two groups of participants, new methods of quantifying differences between individuals are continually being developed, with the goal of measuring precisely those characteristics that are relevant for behavioral investigations (e.g., questionnaire measures of personality, tests of IQ or handedness, genotyping, etc.). The raw materials are therefore readily available for researchers to acquire the data that they need in order to match their groups of items or participants.

However, although the increased availability of tests and norms is crucial if confounding factors are to be controlled, they also introduce a substantial additional problem. How should researchers, faced with several hundred potential items or participants that must be matched on perhaps a dozen different dimensions, select a correctly matched subset for their experiment? As one might imagine, the number of possible solutions to this type of problem can be enormous. For instance, there are more than 10^{15} (i.e., a million billion) ways of selecting 12 control participants from a pool of 100 individuals. For psycholinguistic experiments in which the candidate set of items may only be limited by the number of words known by a typical participant (approximately 50,000) this combinatorial explosion becomes even more severe. Trying to get a good match between two sets of items or participants can be a time-consuming and tedious task, even if the relevant

M. van Casteren, maarten.van-casteren@mrc-cbu.cam.ac.uk

data have already been collected. On the other hand, taking shortcuts and failing to correctly match your experimental conditions will introduce noise into experimental data, reduce the significance of measured effects, or even invalidate experimental results where confounding factors remain uncontrolled.

In this article, we describe a computerized solution to the problem of matching experimental items or participants on multiple, potentially confounding factors. "Match" is a computer program that can automatically select two or more matched groups of stimuli/participants on multiple numerical measures. The program tries to find a solution that will match groups of items or participants as closely as possible in mean, median, and standard deviation values for all the measures involved. Since a precise solution may not be possible in all cases, each measure can be assigned a weight to indicate its relative importance in the matching process. We believe that the use of the Match program will substantially benefit experimental research in a number of different areas of psychology and other disciplines. The factorial designs that can be constructed using this program will ordinarily be more closely matched than those that could be obtained by hand or by using semiautomated approaches (e.g., sorting and resorting the materials, or computing the most similar pairs of items from a larger set; see Lahl & Pietrowsky, 2006). Using Match requires minimal intervention on the part of the experimenter beyond generating a database of candidate items or participants, and writing a script that sets some simple parameters for the matching process.

In addition to the greater convenience and performance of using a computer program for matching sets of items or participants, there is also a more principled reason why a computerized matching process is beneficial, which is that it removes possible sources of experimenter bias. Forster demonstrated the possibility for this in word recognition experiments (Forster, 2000). He tested visual lexical decision (VLD) responses for a list of word pairs that were matched on word frequency. He then asked two groups of 25 volunteers, all of whom conducted research into language processing, to decide which of the pairs of frequency-matched words would produce the faster RT in a VLD experiment. All of the researchers were able to make this decision with better than chance accuracy, irrespective of their degree of research experience (graduate students fared as well as senior researchers). On the basis of this finding, it is therefore possible for a researcher to introduce an experimental effect in a comparison of apparently matched experimental items by judicious stimulus selection. Although it is perhaps unlikely that researchers would deliberately choose items so as to produce a biased experiment, it is clear that subconscious, or unintended selection biases could alter the outcome of psycholinguistic experiments. By using an automated procedure to select the best-matching stimuli, we believe that Match will prevent any such bias and produce better and more reliable results in psycholinguistic experiments.

In this article, we start by giving an example of a matching problem and the results that can be obtained with Match, we then explain the algorithm that is at the heart of the program, and finally, we describe a case study in which we use Match to generate stimuli for an already published EEG (electroencephalography) experiment. We show that this automated approach produces better-matched stimulus sets than those previously generated by hand, and in much less time.

Example

To clarify the use of the program, we have constructed a simple problem to serve as an example. The complete example is also available to download from the Match Web site. In this case, we have two experimental conditions: one containing words with no neighbors, and one with words that have two or more neighbors. A neighbor of a word is another word of equal length that is different by only a single letter. We would like these two conditions to be matched on word length, frequency of occurrence, and bigram frequency. Note that both word length and bigram frequency are correlated with frequency of occurrence, and both frequency measures are correlated as well, making this a nontrivial matching problem.

First, we need two input files with item candidates for each condition. These files should also contain the three dimensions that we want our conditions to be matched on. Table 1 gives an overview of the input to the Match program, listing the first 10 entries for each input file and a summary of the mean values for all dimensions for these two files. As can be seen, the input files contained 1,054 and 962 items. Frequency of occurrence and bigram frequency mean values are already quite closely matched, but the mean word lengths are not. Just selecting words with matching lengths will not work, because this will affect the match on the other dimensions as well, because of the confounds mentioned earlier.

Match was configured to select the best-matching 50 items from each file. The script used to get these results is included as the Appendix. The results are given in Table 2. The summary for these selected files clearly shows that Match has successfully matched all relevant dimensions. The first 10 items from each output file are also included, and these show the way the program has matched items in a pairwise fashion. Each item has a clear match in the other file. This is a result of the algorithm used, which matches items in pairs, or more generally, in tuples, because any number of files can be matched in this way. Items from the same tuple will always be written to the same line number in each output file so that the user can check which items the program grouped together. In this way, a tuple of matched items can also be removed as a whole, if need be.

Algorithm

Match will operate on any number of sets of items or participants and on any number of dimensions. In this description, we will focus on the problem of generating matched sets of experimental items—though the same algorithm operates equivalently when matching sets of experimental participants. Each set of potential items is provided in a separate input file, along with all the necessary dimensions. The goal of the matching process is to select a subset of items from each set, such that these

Table 1
Example of a Matching Problem: Data Sets Before Matching

List 1: No Neighbors				List 2: 2+ Neighbors			
Word	CELEX Frequency	Bigram Frequency	*n*	Word	CELEX Frequency	Bigram Frequency	*n*
abbess	52	36,778	0	abide	84	69,732	2
abortive	59	58,565	0	ably	65	46,160	2
abstain	65	42,429	0	ace	81	103,413	9
abstract	51	36,717	0	ache	86	106,028	3
absurdly	58	28,450	0	adept	59	45,288	2
abyss	72	32,671	0	adorn	75	46,413	2
accrue	72	58,409	0	airy	59	45,558	4
acquit	60	42,741	0	akin	63	56,530	2
activate	96	57,777	0	ale	63	109,706	10
acutely	82	37,210	0	alight	53	44,194	5
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.				.			
Summary		1,054 items		Summary		962 items	
Mean length		6.9 letters		Mean length		4.8 letters	
Mean frequency		69.7 / 17.7 million		Mean frequency		70.0 / 17.7 million	
Mean bigram		31,405 / million		Mean bigram		33,462 / million	

subsets are optimally matched on all relevant dimensions. Matching relationships are specified pairwise between two input sets. More complicated matching is possible by using multiple matching relationships, and there is no limit to the number of matching links. For instance, we will present a case study in which six sets of items needed to be matched, with a total of 12 matching relationships between them.

Match uses a distance measure that reflects the degree of similarity between individual items as the variable to optimize. This distance measure is computed using the normalized standard Euclidean distance between item pairs, calculated over the vector of values on all the dimensions involved in the matching process. These values may be measures such as word frequency or length, in the case of matching items, or age and IQ, in the case of matching participants. Match tries to minimize the sum of the squares of all these Euclidian distances between all sets of matched items.

In operation, Match will first of all read in all the sets of items or participants, along with the values on all of their associated dimensions. These values are then normalized and weighted, if weights were specified. Normalization will ensure that a dimension that happens to have large values will not dominate the matching process at the cost of dimensions with smaller average values. Values for all dimensions that are specified to be matched are normalized to the average mean of each dimension for both input sets in each matching relationship.

Items will be matched in tuples or sets, each containing one item from every input data set. A solution to the matching process is reached when each tuple contains precisely one item from each data set, with each item appearing in only one tuple, because it is not possible for an experimental item or participant to appear more than once in a factorial experiment. In order to generate a solution, we start by creating a set of possible items on each posi-

Table 2
Example of a Matching Problem: Data Sets After Matching

Selection 1: No Neighbors				Selection 2: 2+ Neighbors			
Word	CELEX Frequency	Bigram Frequency	*n*	Word	CELEX Frequency	Bigram Frequency	*n*
abyss	72	32,671	0	shady	72	33,119	2
anoint	69	60,020	0	crease	69	59,951	2
apex	59	30,993	0	wild	59	30,904	9
asthma	70	53,613	0	babble	70	53,340	5
baboon	80	33,501	0	motion	80	33,492	3
basics	52	24,500	0	snatch	52	24,490	2
beggar	66	31,720	0	wicker	66	31,979	8
conifer	78	31,864	0	commend	78	32,288	2
consul	55	24,202	0	wooded	55	23,806	2
effigy	64	16,571	0	rugged	64	16,537	2
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.				.			
Summary		50 items		Summary		50 items	
Mean length		6.2 letters		Mean length		6.2 letters	
Mean frequency		68.0 / 17.7 million		Mean frequency		68.0 / 17.7 million	
Mean bigram		30,983 / million		Mean bigram		30,908 / million	

tion for each tuple. Initially, each of these sets will contain all of the available items in the corresponding input sets.

To reach a solution, the program will have to select the best-matching items in each position within each tuple. After an item has been chosen for a single tuple, the selected item has to be removed from all other tuples, a process called "pruning." The actual matching process consists of an active search for solutions within this pruned search space. The algorithm used in Match will alternate between pruning and selecting, and will backtrack to an earlier state when a search path fails (leaving a tuple without any items) or generates a solution (a complete set of tuples, all of which contain only single items). When the program backtracks, it returns to the last selection choice, and reinstates all the items that were removed from all tuples and then removes the selected item, because this choice has now been exhaustively tried. Each time the program finds a solution, it is compared with the best solution found previously, and if it is better, it will be stored as the current best solution. The criterion used to compare solutions is, again, the total (i.e., summed squared) distance within all tuples. The process of finding the best possible solution to the matching problem is therefore a recursive, depth-first search process through a tree that contains the entire space of possible matches. Since an exhaustive search through this tree is computationally extremely expensive, we can improve efficiency further by adding extra strategies at the selection stage. These will ensure that the algorithm spends most of its time engaged in searches that are likely to lead to a better solution than has currently been saved.

First of all, the choice of best item for the next selection is crucial. But, selecting the best item to be assigned to one position within a tuple is a problem in itself. The quality of match for any given item is completely dependent on the other items that have already been selected within its tuple. When no items have been selected at all, it is rather difficult to decide which item is the best to be selected first, because we need all the other items to calculate the quality of the match. Therefore, selecting the first item is in principle equivalent to selecting a complete solution for the whole tuple. And finding the best-matching tuple with certainty will necessitate going through all possible combinations, a number equal to the product of the sizes of all data sets. In the case study described below, this product was more than 10^{13}, clearly an impossible task. Our solution to this combinatorial explosion is to use a heuristic: a "rule of thumb" that will point the process in approximately the right direction. We start by choosing an item at random from each set; we then loop through all the items in each set and select the one that matches best with all other current choices. Several loops are performed, and the order in which to loop through all sets is randomized too. Making a selection like this for all the initial tuples will ensure that at least one reasonably good solution will be created somewhere. This leaves the problem of which item should be selected first within a tuple. The quality of the match is calculated over the whole tuple, so it will be the same for all items. To see which one is best to select next, we compare the total match for each item within the tuple to the second-best-matching item in the same set. The difference is an indication of how much "matching quality" we would lose if we did not select the best item. The item with the biggest difference for these values is the one that should be selected first.

Second, we can use a simple procedure to decide when a certain search path cannot produce a better solution than has previously been found. We do this by adding up all the squared distances of all the best items, as found by the heuristic described above, in all tuples after pruning. As soon as the best currently possible solution is inferior to the best solution that has previously been found, we can stop searching in this direction, and backtrack. Of course, this will only work correctly if we made the perfect choice for our best-matching items. But since we only used a stochastic "quick and dirty" heuristic, we have to be careful and leave a small margin. This means discarding only search paths that are clearly worse than the current best solution. Currently, this margin is set to 10%.

The algorithm as described will eventually lead to an exhaustive search, in which all possible solutions are explored. As we have suggested at the outset, this can be very time consuming, even for relatively modest problems. However, since a single, guaranteed optimal solution is not ordinarily required, we leave it to the user to decide how long to leave the program running and therefore how good a matching solution is required. The program can be terminated earlier, at which point the best solution found so far will be reported. Since the program always tries to instantiate the best possible domain first, the solution after a limited time will be reasonably optimal. As the following case study illustrates, even when fairly short runs are used, sets of items generated using the Match program are ordinarily superior to those selected by hand.

Case Study

To demonstrate the capabilities of our Match program, we selected a published study in which a complicated matching solution was needed, which had already been solved by hand. The study chosen was "Effects of Word Length and Frequency on the Human Event-Related Potential," by Hauk and Pulvermüller (2004). They performed a VLD experiment to investigate the influence of word length and frequency of occurrence on the visually evoked event-related potential to written words. Their word stimuli were grouped into six conditions, in a 2 × 3 factorial design with length (short vs. long words) and frequency (low-, medium-, and high-frequency words) as independent variables. Words were matched for length in the three frequency conditions, and for both word form and lemma frequency in the two length conditions. Since word length and word frequency are correlated (Zipf, 1949), we can consider this problem of simultaneously matching items on both variables as being rather challenging. In addition to these two variables, Hauk and Pulvermüller also took care to make their conditions as well matched as possible on the standard deviation of these matching properties. They performed an ANOVA on the matched stimulus sets, and could not find any significant differences in length (between the frequency groups), or word form and lemma frequency (for the length groups). They also matched the base 10 logarithm of both frequency measures, and equated

Table 3
Means and Standard Deviations Showing Length Matching From Hauk and Pulvermüller (2004) and As Generated by Match

				Length (Letters)			
				H and P		Match	
Length Conditions	Frequency Conditions	Initial Set Size	Final Set Size	*M*	*SD*	*M*	*SD*
Short	Low	137	69	4.101	0.84	4.101	0.79
	Medium	180	69	4.116	0.87	4.101	0.79
	High	178	69	4.101	0.81	4.101	0.79
Long	Low	113	69	6.275	0.80	6.203	0.81
	Medium	157	69	6.246	0.91	6.203	0.81
	High	153	69	6.246	0.83	6.203	0.81

their words for trigram frequency. However, ANOVAs did reveal some significant differences in orthographic neighborhood size (*N*) for both length and frequency, as well as an interaction between the two. This was attributed to the fact that short, high-frequent words tend to have fewer neighbors in general.

Matching the six conditions was performed by hand by the authors and took a considerable amount of time. Since we had access to the same initial sets of stimuli (from which matched items were chosen) we could apply the Match program to this problem and compare its performance with the matching used in a published study. Table 3 shows the matches of word length for both the original data as well as the Match program. The table also shows the sizes of the initial sets of items, to give an indication of the scale of the matching problem. A matched set of 69 items was to be selected from each of the data sets. As can be seen, Match consistently produced better matches than the human researcher, with regard to both the mean and standard deviation of the critical variables. Indeed, for the lengths, the program found perfect matches by ensuring that only triplets of words with the same number of letters were chosen for the low-, medium-, and high-frequency conditions. Table 4 shows the matching results for the lemma frequency and word form frequency. Match was able to match all conditions within the second decimal, clearly improving on the hand-matched results.

In this case, the results obtained by Match required an overnight run to produce results that clearly outperformed those of a highly motivated human. However, very good matches were obtained after as little as 10 minutes of computing time using a standard Pentium 4, 2.4-MHz desktop PC running Windows 2000. Writing and testing the script took no more than 15 minutes.

In addition to showing that the program could perform the same task as the authors of this article, we explored whether the remaining confound of neighborhood size (*N*) could be removed from this set of items. To this end, additional matching relationships were added to the script and the program was run again. We were not able to completely remove the effect of *N* between the length conditions, but there was no longer any significant main effect of *N* in the comparison of the frequency conditions, nor any interaction between length and frequency. Since both of these effects existed in the previous item set, we would suggest that Match has improved on the previous set of materials. Table 5 shows the results of an ANOVA on the final set of matched stimuli.

Because *N* is very strongly correlated with word length, it is perhaps unsurprising that the program was unable to remove the difference in *N* between words of different lengths. However, within the space of possible matches that exist in the language, Match has produced a clear improvement, in that the effect of *N* was removed from comparisons between frequency conditions. This result proves that Match is not only able to solve the same problems as humans, but can, in some cases, solve problems that are beyond the capabilities of a highly motivated human researcher.

Program

Match is a command-line program that will run on Microsoft Windows 32-bit systems. The program is written in standard C++, so it could possibly be ported to other operating systems as well. There is no graphical user inter-

Table 4
Means and Standard Deviations Showing Lemma and Word Form Frequency Matching From Hauk and Pulvermüller (2004) Compared With the Results As Generated by Match

		Lemma Frequency (log/mil)				Word Form Frequency (log/mil)			
		H and P		Match		H and P		Match	
Frequency Conditions	Length Conditions	*M*	*SD*	*M*	*SD*	*M*	*SD*	*M*	*SD*
Low	Short	0.836	0.266	0.812	0.271	0.666	0.227	0.642	0.228
	Long	0.770	0.302	0.818	0.264	0.604	0.244	0.642	0.228
Medium	Short	1.594	0.238	1.541	0.245	1.422	0.226	1.416	0.232
	Long	1.537	0.284	1.540	0.242	1.412	0.275	1.412	0.237
High	Short	2.240	0.250	2.246	0.232	2.127	0.246	2.140	0.231
	Long	2.247	0.261	2.245	0.231	2.145	0.267	2.141	0.232

Table 5
Statistical Results of an ANOVA Performed on the Second Case Study Matching Exercise

	Property			
Condition	WF Frequency	Lemma Frequency	No. Letters	Neighbors
Length	$F(2,1) = 0.002$ $\boldsymbol{p = .97}$	$F(2,1) = 0.033$ $\boldsymbol{p = .86}$	$F(2,1) = 808$ $\boldsymbol{p = .00}$	$F(2,1) = 335$ $\boldsymbol{p = .00}$
Frequency	$F(3,2) = 345$ $\boldsymbol{p = .00}$	$F(3,2) = 263$ $\boldsymbol{p = .00}$	$F(2,1) = 0.00$ $\boldsymbol{p = 1.0}$	$F(3,2) = 0.001$ $\boldsymbol{p = .999}$

face: All necessary settings and parameters are entered in a very simple script file, which is then passed to Match as a command-line argument. The program can be instructed to run in the background, at a lower priority, to enable running scripts for a longer time without interfering with other activities on the same computer. The program can be stopped at any time, and the best solution at that point can then be written to file. The program can work on the raw values provided with the items, as well as on the logarithm (base 10) or the orthographic length of the input fields. Missing values can be replaced automatically by either a fixed value or the mean of the existing other values. When two selections are being made from the same input file, Match will make sure that each item appears in only a single output file.

Availability

A Match executable for Windows, a complete manual, and a small demo problem can be downloaded from this Web site: www.mrc-cbu.cam.ac.uk/~maarten/match.htm. Use is limited to academic or other nonprofit applications. Terms of use can be found on the Web site mentioned above. Authors who use Match for their research are expected to cite this article.

Conclusion

Match is a program that can not only make the process of matching items for factorial experimental design much easier, it can solve problems that are difficult or impossible to do by hand or with other tools. It will also prevent possible experimenter bias, by automating the stimulus selection process. The case study shows that the results are typically better than those obtained by hand matching, and can be obtained in a fraction of the time, with considerably less effort. The generic nature of the program, and the fact that using Match will increase the power of experiments, makes it an indispensable tool for research in many areas.

AUTHOR NOTE

We thank Olaf Hauk for his help with the case study involving his experiment. Correspondence concerning this article should be addressed to M. van Casteren, MRC Cognition and Brain Sciences Unit, 15 Chaucer Road, Cambridge CB2 7EF, England (e-mail: maarten.van-casteren@mrc-cbu.cam.ac.uk).

REFERENCES

Baayen, R. H., Piepenbrock, R., & Gulikers, L. (1995). *The CELEX Lexical Database* (Release 2) [CD-ROM]. Philadelphia: Linguistic Data Consortium, University of Pennsylvania.

Cutler, A. (1981). Making up materials is a confounded nuisance, or: Will we be able to run any psycholinguistic experiments at all in 1990? *Cognition*, **10**, 65-70.

Forster, K. I. (2000). The potential for experimenter bias effects in word recognition experiments. *Memory & Cognition*, **28**, 1109-1115.

Hauk, O., & Pulvermüller, F. (2004). Effects of word length and frequency on the human ERP. *Clinical Neurophysiology*, **115**, 1090-1103.

Lahl, O., & Pietrowsky, R. (2006). EQUIWORD: A software application for the automatic creation of truly equivalent word lists. *Behavior Research Methods*, **38**, 146-152.

Zipf, G. K. (1949). *Human behaviour and the principle of least-effort*. Cambridge, MA: Addison-Wesley.

APPENDIX
Match Script Used for the Example

```
InputFile F1 LowN.txt Low_sel_3.txt
InputFile F2 HighN.txt HighN_sel_3.txt

MatchFields F1 1 F2 1 UseLength Length
MatchFields F1 2 F2 2 Frequency
MatchFields F1 3 F2 3 Bigram

OutputSize 50

OutputFile ScreenOutput.txt
```

(Manuscript received June 20, 2006;
revision accepted for publication October 9, 2006.)