Behavior Research Methods, Instruments, & Computers
1990, 22 (2), 228-235

SESSION XI
MODELING AND SIMULATION

Scott Gronlund, *Presider*
Northwestern University

Implementation of global memory models with software that does symbolic computation

SCOTT D. GRONLUND, CHING-FAN SHEU, and ROGER RATCLIFF
Northwestern University, Evanston, Illinois

A number of software tools are available that do symbolic computation that can facilitate the theoretical analysis of a mathematical model. We made use of one these tools, Mathematica (Wolfram, 1988), to implement the global familiarity memory models of Gillund and Shiffrin (1984) and Murdock (1982, 1983). We describe our implementation, and illustrate the flexibility and efficiency of use that these tools allow.

Software packages that do symbolic computations have not yet found their place among the tools of researchers in psychology. There are many such tools, including ALTRAN (Brown, 1977), MACSYMA (Rand, 1984), MAPLE (Geddes, Gonnet, & Char, 1983), MATHEMATICA (Wolfram, 1988), muMATH (Wilf, 1982), REDUCE (Hearns, 1973), and SCRATCHPAD (Griesmer, Jenks, & Yun, 1975). Symbolic computation differs from numerical computation, in that it results in an expression in terms of the parameters of a model; numerical computation requires that the parameters of a model be given particular numerical values. The symbolic expression for a model is preferable; it is more general, and it allows relationships among the model's parameters to be examined. We will demonstrate the potential of these symbolic computation tools by illustrating our work implementing the global memory models of Gillund and Shiffrin (1984) and Murdock (1982, 1983), using Mathematica (Wolfram, 1988). An implementation of this sort facilitates the derivations and encourages more extensive exploration of the model space. We will begin with a sample derivation for both models for a paired-recognition task. This will help one understand the Mathematica implementation that follows.

This research was supported by NSF Grant BNS 85-10361 and NIMH Grant MH 44640 to Roger Ratcliff, and NSF Grant 85-16350 to Gail McKoon. Correspondence concerning this article should be addressed to Scott D. Gronlund, who is now in the Department of Psychology at the University of Oklahoma, Norman, OK 73019.

GLOBAL FAMILIARITY MODELS

An important paradigm for evaluating global familiarity models is the paired-recognition paradigm (see, e.g., Gronlund & Ratcliff, 1989; Humphreys, Pike, Bain, & Tehan, 1989), in which subjects study n pairs of words sequentially at some rate. In the test phase, the subjects might be asked to determine whether the test words have or have not been studied previously.

The models of Gillund and Shiffrin (1984) and Murdock (1982, 1983) base a recognition decision on the interaction of a test probe (the word) with all items in memory. (The MINERVA 2 model of Hintzman [1984, 1988] works similarly; an analytic treatment of this model can be found in Sheu [1990].) The result of this interaction is a measure of global matching or strength: The more similar the test probe is to one or many traces in memory, the greater the matching strength is. The greater the matching strength, the more "familiar" the test probe. A test word from a study pair generally has a greater matching strength than an unstudied word does, because a test word from a study pair would match two items in memory, whereas an unstudied word would match none.

A test word from a study pair is called a *target*, an unstudied test word a *distractor*. The global memory models assume that the strengths of targets and distractors vary due to rehearsal, differences among words, and so forth, resulting in a distribution for target and for distractor familiarity. Therefore, these models can be cast in a signal detection framework, and derivations can be computed in terms of d'. Thus, the mean and variance of the target and distractor distributions need to be computed.

The SAM Model

In the SAM model of Gillund and Shiffrin (1984), memory is represented as a matrix of strengths of connection between possible retrieval cues ($Q_1, Q_2, \ldots Q_m$) and stored traces or "images" ($I_1, I_2, \ldots I_n$). The retrieval strength between a cue and an image is given by $S(Q_j, I_i)$. (Retrieval strength is derivative of the strength stored during learning, but reflects what is available at retrieval. A relatively large strength of connection indicates that the cue is a good cue for that image; e.g., *cat* as a retrieval cue for *dog*.)

Recognition based on global familiarity is the product of the strengths of connection of the cues ($Q_1, \ldots Q_m$) to an image (I_i), summed over all the images in memory. The cues can be differentially weighted (w_j) to reflect differing amounts of attention to a cue at retrieval. The sum of the w_js is 1.0 in a limited-capacity retrieval system. The computation of familiarity is summarized by Equation 1 (Gillund & Shiffrin, 1984):

$$F(I_1, I_2, \ldots I_n; Q_1, Q_2, \ldots Q_m) = \sum_{i=1}^{n} \prod_{j=1}^{m} S(Q_j, I_i)^{w_j}. \quad (1)$$

The SAM model has been applied mostly to episodic tasks (see Gronlund & Shiffrin, 1986, for an application to semantic tasks). In an episodic task, the strengths connecting cues to images, $S(Q_j, I_i)$, are assumed to be a function of a buffer rehearsal process (see, e.g., Atkinson & Shiffrin, 1968). Three parameters govern the strengths stored as a function of rehearsal: (1) a units of strength are stored per unit of rehearsal time between context and an image (*context* reflects the situational, emotional, and physical environment present at the time of learning); (2) b units of strength are stored per unit of rehearsal time between two images that occupy the rehearsal buffer together (interitem strength); (3) c units of strength accrue per unit of rehearsal time to an image resident in the buffer (self-strength). A fourth parameter (d) reflects residual strength between two images never rehearsed together (pre-experimental factors).

The variability in strengths that result from rehearsal must be supplemented by the addition of a noise component. (This could reflect neural noise.) Otherwise, the familiarity of a target would always exceed that of a distractor because $b > d$, and recognition performance would be perfect. To incorporate this noise component, Gillund and Shiffrin (1984) assumed that the strength available at retrieval $S(Q_j, I_i)$ is a random variable with the following three-point distribution:

$$\begin{aligned} (1+v)\mu &= \tfrac{1}{3} \\ \mu &= \tfrac{1}{3} \\ (1-v)\mu &= \tfrac{1}{3}, \end{aligned} \quad (2)$$

where v specifies the divergence of the distribution. Gillund and Shiffrin (1984) assumed that this noise is independent across cues and images.

Pair-recognition derivation. Assume that subjects study J pairs of words sequentially for t seconds per pair. We will compute the derivation for when the discrimination is between a single word from a studied pair (target) and an unstudied word (distractor).

To facilitate derivations, two simplifying assumptions are made (Clark & Shiffrin, 1987). First, the v parameter for the spread of the noise distribution, and the attention weight given to the context cue (w_c, with the item cues sharing equally the remaining weight), must be numerical. The value 0.5 is commonly chosen for both. This means that the expected value of the retrieval strength after the addition of the noise is $\mu^{w_c}/3\ [(1+v)^{w_c}+1+(1-v)^{w_c}]$. Call this quantity $k\mu$. Its second moment is given by $\mu^{2w_c}/3\ [(1+v)^{2w_c}+1+(1-v)^{2w_c}] = \mu$.

Second, encoding is strictly controlled; rehearsal is limited to be within-pair only (d strengths elsewhere). This is not unreasonable if subjects have to discriminate test pairs that exactly match a study pair from test pairs that consist of two words that were studied, but not together

Table 1
SAM Model Retrieval Matrix with Within-Pair Rehearsal Only

		Traces						
		A1	B1	A2	B2	...	AI	BI
Probe→	A1	$ct+d$	$bt+d$	d	d		$S(Q_j,I_i)$	d
	B1	$bt+d$	$ct+d$	d	d		d	d
	.							
	.							
	.							
	AJ	d	d	d	d		$ct+d$	$bt+d$
	BJ	d	d	d	d		$bt+d$	$ct+d$
Probe→	context	at	at	at	at		at	at
	new	d	d	d	d		d	d

Note—If A1 and context are used as the probe cue, the self-strength component is $k(ct+d)^{.5}k(at)^{.5}$, the interitem component is $k(bt+d)^{.5}k(at)^{.5}$, and the residual component is $kd^{.5}k(at)^{.5}$. There are $(2I-2)$ residual strengths total. The expected familiarity of a target is the sum of these components. The expected strength of a distractor (row *new* and context) is $2I(kd^{.5}k(at)^{.5})$. The variance of a *new* and context cue's connection to a single trace is $(at)d-[kd^{.5}k(at)^{.5}]^2$, which is the second moment minus the first moment squared. Therefore, the standard deviation of all $2I$ distractors is $[2I\ atd(1-k^4)]^{.5}$, and d' is $k^2/(1-k^4)^{.5}1/(2I)^{.5}[(ct/d+1)^{.5} + (bt/d+1)^{.5}-2]$.

(see, e.g., Gronlund & Ratcliff, 1989). The resulting retrieval matrix is given in Table 1.

In an episodic task, it is assumed that the subject uses the context cue to focus on the images from the appropriate list. To accomplish this, the context cue is used together with the item cue in a joint probe of memory, each cue getting 0.5 attention weight. The familiarity of a test probe consisting of a studied word and the context cue contains three components: (1) the self-strength representing the strength of connection between a cue and its own image, $(ct+d)^{0.5}at^{0.5}$, in the first column of Table 1; (2) the interitem strength representing the strength of connection between two items rehearsed together, $(bt+d)^{0.5}at^{0.5}$, in the second column; and (3) residual strength for two items not rehearsed together, $d^{0.5}at^{0.5}$, in columns 3 through $2I$.

Noise (k) as given by Equation 2 is attached to each retrieval strength. The global familiarity of a studied word is the sum of the one self-strength component, the one interitem component, and the $2I-2$ residual components. The component familiarities are given at the bottom of Table 1. The computation is similar for the distractor expectation using the context cue and an unstudied word (row *new*).

To compute the variance for a particular trace when *new* is the cue, the second moment is computed, and the first moment squared is subtracted from it. The overall variance for the distractor is the sum of these individual variances. This quantity is also given in Table 1. We will see that our automated Mathematica implementation will produce these same quantities, while eliminating the need to make v, w_c numerical, and limiting rehearsal to within-pair only. Before this is described, however, the derivation procedure for TODAM will be described.

The TODAM Model

TODAM (Murdock, 1982) assumes a vector representation for memory traces. Items are represented by independent random vectors (e.g., **f** and **g**), which are vectors whose elements are random variables. These random variables are drawn from a normal distribution with mean zero and variance $1/N$. All these random variables are independent and identically distributed.

A single, common memory vector **M** stores information about individual words (item information), as well as information about pairs of words (associative information). To store item information, the item vector **f** is added to **M**. To learn a pair of words, the individual item vectors (**f** and **g**) are added to **M**, as well as the convolution of the two vectors (**f***g) (see Eich, 1982; Murdock, 1982).

Encoding need not be perfect. In the SAM model, this is handled by assuming that t units of strength are stored per unit of rehearsal time. (The longer an item is rehearsed, or the more times it is repeated, the stronger its retrieval strength.) In TODAM, imperfect or probabilistic encoding is governed by a function p (where $0.0 \leq p \leq 1.0$), which governs the probability that a particular element is encoded correctly or encoded as zero. The greater p, the better is the encoding of the vector. (See Lewandowsky & Murdock [1989] for a different method of probabilistic encoding.)

At encoding, different types of information can be weighted to differentially focus attention on associative or item information. These parameters are γ_1 and γ_2 for item information, and ω for associative ($\gamma_1 + \gamma_2 + \omega = 1.0$). To model forgetting, the current vector **M** is discounted by $\alpha(\alpha < 1.0)$ as each additional word set (item vectors and convolution) is added to memory. The storage of the jth pair in the memory vector **M** is given by: $\mathbf{M}_j = \alpha\mathbf{M}_{j-1} + \gamma_1\mathbf{f}_j + \gamma_2\mathbf{g}_j + \omega(\mathbf{f}_j*\mathbf{g}_j)$.

At retrieval, information can be differentially weighted as the result of (e.g.) decision rules that emphasize associative over item information (see, e.g., Gronlund & Ratcliff, 1989). These parameters are denoted r_a for the retrieval weight on the associative information, and r_i for the retrieval weight on the item information. They are free to vary between 0 and 1.

A recognition decision is based on the dot product of the probe vector with the memory vector **M**. The more similar the retrieval probe is to one or more traces making up the memory vector, the greater is the dot product (and the more "familiar" is the probe). The resulting value of the dot product (the equivalent of familiarity in the SAM model) is fed into a two-criterion decision system (Hockley & Murdock, 1987; see also Gronlund & Ratcliff, in press). In pair recognition, the recognition decision is based on the sum of the dot products of the two item vectors plus the dot product of the convolution.

To derive d' requires the expectation and variance of the dot-product similarity between the probe vector and the memory vector for studied (old) and unstudied (new) items. Weber (1988) provides tables of component variances that facilitate these derivations. (We will describe below how most of this work could have been done with symbolic software. In some cases there is an improvement in accuracy.)

Pair recognition derivation. Assume $2J$ word pairs presented for t seconds each. (Presentation time would be reflected in the value of the p function.) The memory vector **M** contains the 2**J** individual vectors, and **J** convolution vectors. In the test phase, discrimination is between a single word from a studied pair (target) and an unstudied word (distractor). To simplify the derivation, set the retrieval weights r_a and r_i to 1.0.

The expectation for a probe consisting of a studied word is p; the expectation for a probe consisting of an unstudied word is 0 (see Weber, 1988). The variance of a studied probe receives contributions from four sources contained within the memory vector: (1) The probe matches an item vector that had been stored in the memory vector, $2/N$; (2) the probe matches no item vector, $1/N$; (3) there is a partial overlap of the probe vector with a convolution vector $(7N^2+4N+1)/(4N^3)$; (4) there is no overlap between the probe vector and a convolution vector $(3N^2+1)/(4N^3)$. The old-item probe variance is equal to the sum

of these terms (counting the correct number of each term), with the appropriate weighting parameters (γ_1, γ_2, and ω) attached.

The serial positions of these terms is relevant in TODAM because α depends on serial position. Murdock (in press) and Murdock and Lamon (1988) assume a fixed serial order to simplify derivations. For purposes of simplification, assume that **J** equals 2, and that the matching item vector was from the second of the two pairs studied. The old probe variance is then given by $\gamma_1 2/N + \gamma_1 1/N + 2\alpha[\gamma_2(1/N)] + \omega(7N^2+4N+1)/(4N^3) + \alpha\omega(3N^2+1)/(4N^3)$. The derivation for the unstudied probe variance is similar. It is straightforward to calculate d' from these component expressions.

This completes a description of how derivations are carried out in the SAM and TODAM models for a single item tested in a pair-recognition paradigm. In addition to relieving the researcher of tedious, time-consuming, and error-prone algebraic manipulations, an automated implementation can facilitate and supplement these derivations in several different ways. We will discuss these as we describe the Mathematica implementation of SAM and TODAM.

MATHEMATICA IMPLEMENTATION

The SAM Model

To implement the SAM model, we made use of Mathematica's programming capabilities. (The language is similar to C, but anyone with knowledge of a computer language would have little trouble learning Mathematica's syntax.) We divided the derivation into three steps: (1) Produce the retrieval matrix, (2) select the probe cues; and (3) compute familiarity and d'.

The first step involves specifying the retrieval matrix according to the experimental design. This is executed by a function called "study" that takes 7 arguments. Four of the arguments are the encoding parameters of the model: a, b, c, and d (a represents context strength, b represents interitem strength, c is self-strength, and d reflects residual strengths). The other three arguments reflect the experimental design; these are the number of words simultaneously studied (*mtuple* = 2 for pairs, 3 for triples, etc.), how many mtuples are studied (the list length or *ll*), and the presentation time (*time*). All arguments except ll and mtuple may be symbolic.

"Study" creates the retrieval matrix for the particular experimental paradigm, and fills in the retrieval strengths. As a first step, we assumed, as did Clark and Shiffrin (1987), that rehearsal is only within-, not between-mtuples. (Alternative encoding strategies are discussed below.) The resulting memory matrix is a $2J \times 2J + 2$ (images $\times$ cues) matrix, where each studied word is encoded as an image and as a potential retrieval cue (as illustrated in Table 1). The extra two rows are for the context cue, and for an unstudied distractor used as a cue. The columns give that cue's connection to each image in memory.

The next step is to specify the probe set by specifying the appropriate rows (cues) of the retrieval matrix. This is done interactively by the function "cues." "Cues" takes three arguments, all numeric: (1) ll; (2) mtuple; and (3) the number of cues in the probe set (*nq*). Any combination of words can make up a probe set. For example, a distractor probe might consist of rows (cues) $2J + 1$ (context) and $2J + 2$ (unstudied item).

The final step in the derivation is the "momentfam" function, which computes the expectation and variance. "Momentfam" embodies Equation 1; it takes four arguments. One argument is the number of columns (number of images) in the memory matrix (*npres*, usually ll $\times$ mtuple). The second argument is the number of cues (nq). The final two arguments may be symbolic, and they are the parameters v (size of the step in Equation 2), and w_c (attention weight on the context cue). "Cues" and "momentfam" must be executed twice, once for the target and once for the distractor. A "dprime" function takes the difference between the two expectations, divides it by the noise standard deviation, and simplifies the expression. It is a straightforward exercise to make the derivation more general by manipulating the d' expression so that ll is represented symbolically. Partially symbolic results, or numerical results, can be achieved by giving explicit values to some or all of the parameters. An annotated version of the "study," "cues," and "momentfam" programs are given in the Appendix.

There are several advantages of this automated implementation over the standard method of doing derivations. One advantage is that the same set of functions can produce derivations for many other experimental designs. For example, suppose there are 10 3-tuples studied at a 5-sec rate, and we wish to compute d' for the following test probes: The target consists of the context cue and three words that had been studied together; the distractor consists of the context cue and two words studied together, plus one word that was studied, but not with the other two. Again, limit rehearsal to be within-3-tuples. The arguments for "study" are 10, 3, 5, a, b, c, d (10 is ll, 3 is mtuple, 5 is time). Next, pick the appropriate cues with "cues." For "momentfam," the arguments are 30, 4, 0.5, 0.5 (30 is npres, 4 is nq, 0.5 is v or noise, and the last 0.5 is for w_c or attention weight on the context cue).

Computation of d' for another target and distractor set for this experimental design only requires redoing "cues" and "momentfam." Without this implementation, every change would require going through the derivation procedure described above. And this implementation is flexible enough so that other paradigms (e.g., the mixed/pure design of Ratcliff, Clark, & Shiffrin, in press) can be accommodated with little or no modification to the original programs.

A second advantage of an automated implementation is that it can allow a model to be more fully explored. Recall that two assumptions were made to simplify the derivations. For one, v and w_c were fixed at 0.5. An automated derivation procedure can produce d' derivations with different values of v and w_c, or can even treat these parameters symbolically (though the resulting expressions are very long). Without an automated implementation,

freeing up these parameters greatly increases the complexity of the derivations.

The value of v (noise) will affect the variance of the target and distractor distributions. Recent work by Ratcliff et al. (in press) points to the importance of variance in the evaluation of global familiarity models. Freeing up this parameter may allow the SAM model to address these data in a way different from that suggested by Shiffrin, Ratcliff, and Clark (in press).

Most importantly, an automated implementation removes the restriction of within-mtuple rehearsal (Clark & Shiffrin, 1987). More realistic encoding assumptions can be instantiated in "study" to produce across-mtuple rehearsal (with minor programming changes). The "cues" and "momentfam" programs operate unchanged on the retrieval matrix created.

There are several rehearsal encoding options to consider: (1) the size of the rehearsal buffer (usually equal to four); (2) how new to-be-learned items replace existing items in the buffer once the capacity of the buffer is exceeded (randomly, or is the longest resident item replaced by the new item?); (3) whether or not the amount of rehearsal devoted to an item is a function of the number of other items currently in the buffer. In a buffer encoding scheme, retrieval strengths vary by serial position (i.e., how many other items are in the buffer, or how long an item remains in the buffer). To compute d' for several serial positions would be formidable without an automated implementation. By incorporating these more realistic encoding strategies, the model can address serial position data, lag effects, and related phenomena, in an analytic manner.

A problem with using random encoding strategies is that a very large number of possible rehearsal patterns could result (e.g., the fourth item might remain in the buffer throughout the list, it could be replaced by the fifth item, etc.). Although one could specify the probability distribution for how long an item remains in the buffer, and how often it was rehearsed with what items, it would not be easy. How can a "typical" retrieval matrix (i.e., average pattern of rehearsals) be approximated for a particular set of rehearsal buffer assumptions? An automated implementation affords the following solution.

A modified version of "study" with a random rehearsal buffer could be executed a number of times with different random seeds. Expectations and variances could be computed for each different retrieval matrix that is produced. These expectations and variances would be accumulated, and then averaged. The average expectation and variance would then be used to compute d'. This would reflect d' for a typical retrieval matrix resulting from these rehearsal buffer assumptions. This has some of the properties of a Monte Carlo simulation, but each "simulated" expectation and variance is exact for a given rehearsal pattern.

The TODAM Model

So far, our implementation of the TODAM model is less general than the SAM implementation. Whereas a single set of programs can generate practically any derivation for SAM, the Mathematica implementation of TODAM is (at this point) not as fully automated. We can do most of the same things, but more modification to programs is required. Nevertheless, the automated TODAM implementation is more efficient and less error-prone than doing the derivations by hand. The TODAM implementation has three components: (1) reading in a table of the appropriate component variances; (2) selecting the target and distractor probe set; and (3) computing the dot product match and variance.

The first step in the automated derivation is to read in the appropriate table of component variances. These depend on the specifics of the experiment. For example, a different table of component variances is required if the mtuple size is changed, or if, in addition to the storage of item vectors and pair-wise convolutions, autoassociative vectors (Eich, 1982) or triple convolutions are stored. The program must be modified to count the correct number of matches, partial matches, and nonmatches. Reading in the component variances is more efficient than computing them "on-line," because their computation is very time-consuming (see discussion below). The uniqueness of each variance computation in the TODAM model is the reason why its implementation is not as general as the SAM model implementation. Weber (1988) provides a subset of component variances that can be used to construct these tables.

The second step in the automated derivation involves the selection of any target or distractor probe set by a "cues" function. The final step is the computation of the dot-product match and variance for this probe set. For the pair-recognition paradigm described above (with no across-pair rehearsal), the program that computes the dot-product expectation and the variance takes nine arguments, two of which (ll and nq) must be numeric. The remaining arguments are the parameters of the model α (forgetting), p (probabilistic encoding), γ_1, γ_2, ω (encoding weights on Item 1, Item 2, and associative information), and r_a, r_i (retrieval weights). The program computes the dot product, sums up the appropriate component variances for a given probe, and computes the overall variance.

Despite being less general than the SAM implementation, there are several advantages to the current TODAM implementation. One advantage is that the same set of functions can produce derivations for other experimental designs, or other targets and distractors (although a different table of component variances may be required). A second advantage of the automated implementation is that it does facilitate exploration of the model. For example, encoding assumptions could be relaxed, allowing across-mtuple convolution vectors to be stored. As an alternative to fixing the presentation order (and simplifying the effect of α, as did Murdock & Lamon, 1988), derivations could be produced for targets at different serial positions. This allows the model to more readily address serial position data, lag effects, and related phenomena.

The problem with relaxing encoding assumptions (e.g., interfacing a buffer encoding process) is that vector com-

binations may be stored that require component variances not in the table of variances (i.e., not in Weber, 1988). For example, if the buffer size is 3 or 4, triple and quadruple convolutions must be stored. Weber (1988) contains the component variances necessary to apply the model to a range of single- and pair-recognition paradigms, but provides only a subset of the expressions necessary to apply the model to situations involving 3-tuples, and none involving 4-tuples. To compute these additional expressions would be difficult and extremely tedious.

However, we were able to automate a procedure for computing these expressions. First, a set of expectation rules must be applied to each element of the dot product of the probe with the memory vector. (Recall that each element in a vector is a normal random variable with mean zero and variance $1/N$, and all these random variables are independent and identically distributed.) This is done for several vector lengths (N), and a polynomial function is fit to the resulting values. The solution of the polynomial function gives the coefficients in terms of powers of N for a given component variance. Although this is too time-consuming to do "on-line" (an hour on a DECstation 3100), it is orders of magnitude easier and more pleasant than doing it by hand. These component variances need only be computed once, and they can then be added to the tables of component variances. Automating this procedure opens up the possibility of computing component variances for a greater variety of probes, and thereby extending the domain of the TODAM model. However, our current methods of computation produce expressions with an enormous number of terms such that it is infeasible for triple convolutions and beyond. (Actually, our current method can compute the variance for a single vector matched against a triple convolution, but not a pairwise or triple convolution matched against a triple convolution.) Nevertheless, there are ways in which these methods could be improved and this problem circumvented. For example, the computation can be done in pieces, a small segment of the memory vector at a time.

Pros and Cons of an Automated Implementation

There are several advantages to implementing a mathematical model in a computational system (Mathematica) in the way we have illustrated for the global memory models of Gillund and Shiffrin (1984) and Murdock (1982). The most obvious advantages are the time saved in doing the derivation, and the increased accuracy of the results (assuming the automated implementation has been debugged properly). On the theoretical side, assumptions that were made to simplify derivations can be relaxed. For example, in the SAM model, $v = 0.5$ and $w_c = 0.5$. It is easy to explore the effect of changing the value of v or w_c, or to let these parameters be symbolic. More realistic encoding assumptions can also be explored in SAM and TODAM. Relaxing these assumptions put the models in touch with additional data.

One must be mindful of a disadvantage as well. It is easy to be seduced into thinking that because a computer provides the answer, it must be correct. This is especially problematic, given how quickly one can produce complicated derivations that are not easily verified. In this case, a tool like Mathematica can serve a different function, by facilitating the checking process: Parameters can be set equal, and simplified versions of the model and extreme cases can be examined.

An automated implementation is a powerful tool for exploring a model. The process of doing derivations is often tedious and seldom enlightening. Symbolic software packages, however, offer the opportunity for the scientist to shift his or her time and effort from the tedium of computation to the more important and interesting exploration of theory.

REFERENCES

ATKINSON, R. C., & SHIFFRIN, R. M. (1968). Human memory: A proposed system and its control processes. In K. W. Spence & J. T. Spence (Eds.), *The psychology of learning and motivation* (Vol. 2, pp. 89-105). New York: Academic Press.

BROWN, W. S. (1977). *ALTRAN user's manual* (4th ed.). Murray Hill, NJ: Bell Laboratories.

CLARK, S. E., & SHIFFRIN, R. M. (1987). Recognition of multiple-item probes. *Memory & Cognition*, **15**, 367-378.

EICH, J. M. (1982). A composite holographic associative recall model. *Psychological Review*, **89**, 627-661.

GEDDES, K. O., GONNET, G. H., & CHAR, B. W. (1983). *MAPLE user's manual*. Waterloo, ON, Canada: University of Waterloo.

GILLUND, G., & SHIFFRIN, R. M. (1984). A retrieval model for both recognition and recall. *Psychological Review*, **91**, 1-67.

GRIESMER, J. H., JENKS, R. D., & YUN, Y. Y. (1975). *SCRATCHPAD user's manual* (Rep. RA70). Yorktown Heights, NY: IBM Watson Research Center.

GRONLUND, S. D., & RATCLIFF, R. (1989). Time course of item and associative information: Implications for global memory models. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **15**, 846-858.

GRONLUND, S. D., & RATCLIFF, R. (in press). Analysis of the Hockley and Murdock decision model. *Journal of Mathematical Psychology*.

GRONLUND, S. D., & SHIFFRIN, R. M. (1986). Retrieval strategies in recall of natural categories and categorized lists. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **12**, 550-561.

HEARNS, A. C. (1973). *REDUCE 2 user's manual* (2nd ed.). (Rep. UCP-19). Salt Lake City, UT: University of Utah Computational Physics Group.

HINTZMAN, D. L. (1984). MINERVA 2: A simulation model of human memory. *Behavior Research Methods, Instruments, & Computers*, **16**, 96-101.

HINTZMAN, D. L. (1988). Judgments of frequency and recognition memory in multiple-trace memory model. *Psychological Review*, **95**, 528-557.

HOCKLEY, W. E., & MURDOCK, B. B., JR. (1987). A decision model for accuracy and response latency in recognition memory. *Psychological Review*, **94**, 341-358.

HUMPHREYS, M. S., PIKE, R., BAIN, J. D., & TEHAN, G. (1989). Global matching: A comparison of the SAM, MINERVA II, Matrix, and TODAM models. *Journal of Mathematical Psychology*, **33**, 36-67.

LEWANDOWSKY, S., & MURDOCK, B. B., JR. (1989). Memory for serial order. *Psychological Review*, **96**, 25-57.

MURDOCK, B. B., JR. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review*, **89**, 609-626.

MURDOCK, B. B., JR. (1983). A distributed memory model for serial-order information. *Psychological Review*, **90**, 316-338.

MURDOCK, B. B., JR. (in press). Learning in a distributed memory model. In C. Izawa (Ed.), *Current issues in cognitive processes: The Flowerree Symposium on Cognition*. Hillsdale, NJ: Erlbaum.

MURDOCK, B. B., JR., & LAMON, M. (1988). The replacement effect: Repeating some items while replacing others. *Memory & Cognition*, **16**, 91-101.

RAND, R. H. (1984). *Computer algebra in applied mathematics: An introduction to MACSYMA*. Boston: Pitman.

RATCLIFF, R., CLARK, S. E., & SHIFFRIN, R. M. (in press). The list-strength effect: I. Data and discussion. *Journal of Experimental Psychology: Learning, Memory, & Cognition*.

SHEU, C.-F. (1990). *A note on the multiple-trace model without simulation*. Manuscript submitted for publication.

Shiffrin, R. M., Ratcliff, R., & Clark, S. E. (in press). The list-strength effect: II. Theoretical mechanisms. *Journal of Experimental Psychology: Learning, Memory, & Cognition*.

WEBER, E. U. (1988). Expectation and variance of item resemblance distributions in a convolution-correlation model of distributed memory. *Journal of Mathematical Psychology*, **32**, 1-43.

WILF, H. S. (1982). The disk with the college education. *American Mathematics Monthly*, **89**, 4-8.

WOLFRAM, S. (1988). *Mathematica: A system for doing mathematics by computer*. Redwood City, CA: Addison-Wesley.

APPENDIX
"Study," "Cues," and "Momentfam" Programs

Note—The noise function implements Equation 2. Dprimesam computes d'. Power0 makes $x^{0.0}$ equal to x^0 equal to 1.0.

Study function

```
study[ll_,mtuple_,time_,a_,b_,c_,d_,dc_]:=
    Block[{i,j,k,l,m,n,o,z,p},
        rmat=Table[Switch[i-j,0,dc,_,d],{i,(ll mtuple) + 2},{j,ll mtuple}];
(*
creates retrieval matrix, rmat, and fills w/ residual=d for interitem and dc for self strengths, most
applications dc = d
*)
        z=0;
(*
rehearsal of ll mtuples, no across-mtuple rehearsal
*)
        For[k=1,k<=ll,k++,(*add b and c strengths within-mtuple*)
            For[l=1,l<=mtuple,l++,
        z=z+1;
                For[m=1,m<=mtuple,m++,
                    If[(m + (k-1)) != (l + (k-1)),
                        rmat[[z,m+mtuple (k-1)]]=rmat[[z,m+mtuple (k-1)]]+b time,
                        rmat[[z,m+mtuple (k-1)]]=rmat[[z,m+mtuple (k-1)]]+c time
                    ];
                ];
            ];
        ];
(*
add context strengths at (ll + 1,1), new item located at (ll + 2,1)
*)
        For[p=1,p<=(ll mtuple),p++,
            rmat[[(ll mtuple) + 1,p]] = a time;
        ];
(*
build matrix qmat used by cues function, qmat contains correspondences between the cues chosen to probe
with, and the rows of rmat
*)
        qmat=Table[o + (mtuple (n-1)),{n,ll+2},{o,mtuple}];
        qmat[[ll+2,1]]=qmat[[ll+1,1]]+1;
    ]
```

Cues function

```
cues[ll_,mtuple_,nq_]:=
    Block[{p},
(* function assumes that the context cue is being used, and that it will be entered first*)
    qll[1]=Input["which mtuple? (context cue at LL + 1)"];
    qm[1]=Input["which word in the mtuple? (context cue at 1)"];
```

APPENDIX (Continued)

```
    For[p=2,p<=nq,++p,
        qll[p]=Input["which mtuple? (new word at LL +2)"];
        qm[p]=Input["which word in the mtuple? (new word at 1)"];
     ];
(*qll and qm used in conjunction with qmat to select proper rows (cues) from rmat*)
   ]
```

Momentfam function

```
momentfam[npres_,nq_,v_,wc_]:=
      Block[{i,j,k,jj},
(*
implementation of Equation 1 from Gillund & Shiffrin, 1984
*)
              For[j=1,j<=npres,j++,(*expectation computation*)
                  For[i=2,i<=nq,i++,
                              prod[i]=rmat[[qmat[[qll[i],qm[i]]],j]]^((1-wc)/(nq-1)) noise[(1-wc)/(nq-1),v];
                  ];
(*
context cue
*)
                  prod[1]=rmat[[qmat[[qll[1],qm[1]]],j]]^wc noise[wc,v];
(*
cues' connection to trace j
*)
                  tmpprod[j]=Product[prod[k],{k,1,nq}];
              ];
(*
sum up cues' connection to all npres traces
*)
              meansum=Sum[tmpprod[jj],{jj,1,npres}];(*expectation*)
(*
second moment computation
*)
              For[j=1,j<=npres,j++,
                   For[i=2,i<=nq,i++,
                           prod1[i]=rmat[[qmat[[qll[i],qm[i]]],j]]^(2(1-wc)/(nq-1)) noise[2(1-wc)/(nq-1),v];
                   ];
                   prod1[1]=rmat[[qmat[[qll[1],qm[1]]],j]]^(2 wc) noise[(2 wc),v];
                   tmpprod1[j]=Product[prod1[k],{k,1,nq}];
              ];
              variance=Sum[(tmpprod1[m1]-tmpprod[m1]^2),{m1,1,npres}]/npres;
              Return[meansum,variance];
      ]
```

Additional required functions.

```
1. noise[w_,v_]:=((1-v)^w)/3. + (1./3.) + ((1+v)^w)/3.

2. dprimesam[mean1_,mean2_,var2_]:=Expand[(mean1-mean2)/Sqrt[var2]]

3. power0:=(Unprotect[Power]; Power[ x_ , 0.]:= 1; Protect[Power])
```