Imagine you are a medical doctor. How comfortable would you be prescribing a newly developed medication for which there was minimal knowledge of how it works or of its side effects? An analogous situation exists today in the psychological sciences in modeling human behavior. Quantitative modeling has evolved into an influential and increasingly popular research tool and method of inquiry, yet our understanding of model behavior is too often quite limited (e.g., why the model behaves in a particular way and the range of other behaviors it exhibits). In fact, model behavior can be down right mysterious (Dawson & Shamanski, 1994; McCloskey, 1991). This observation is not a criticism about models or modeling, but a comment about the absence of methods for studying them.

Modelers need to understand the consequences of their design choices in model construction. They also need to be able to make well-informed decisions when choosing between competing models. The computational power and precision of quantitative models of human behavior requires correspondingly sophisticated analysis tools. The purpose of this paper is to introduce and demonstrate such a tool.

Current Methods of Model Analysis

Much of present-day model evaluation and comparison assess what we have recently dubbed Local Model Analysis (LMA; Navarro, Pitt, & Myung, in press). The term refers to evaluation in a very specific context The most frequent form of LMA is testing model performance against human performance, for example, by fitting a mathematical model to the data or simulating the human pattern with a connectionist model. Because data are a reflection of the psychological process underlie study, a good fit to the data is a necessary condition a model must satisfy to be taken seriously. A good fit determines how well a model passes the sufficiency test of mimicking human performance. It is especially useful in the early stages of model development as a quick and easy check on sufficiency. Quantitative measures of fit include percent variance accounted for, root mean square deviation, and maximum likelihood ().

As with any quantitative measure, it is important to be aware of the limitations of goodness-of-fit (GOF) measures. For example, they can be biased when assumptions are violated (normality in the case of root mean square deviation). What we can learn from them is also limited. A good fit makes a model a member of the class of possible contenders. The problem is that this class will almost always be large. If two models that one is comparing fit the data similarly well, other analysis methods are needed to choose between them.

Another LMA method that is useful for probing model behavior more deeply is a sensitivity analysis, in which a model's parameters are varied around its best-fitting values to learn how robust model behavior is to slight variations in those parameters. If a good fit reflects a fundamental property of the model, then this behavior should be stable across reasonable variation in the relevant parameter(s). Another reason a model should satisfy this criterion is that human data are noisy. A model should not be so sensitive that its behavior changes noticeably when noise is encountered. Cross validation, in which a model is fit to the second of two data sets using the best fitting parameter values from fitting the first data set, is a fit-based approach to quantifying this sensitivity.

A drawback of all LMA methods is that they are local. For example, each fit provides a view of model performance in a specific experimental setting or (e.g., design). Each view is also always in relation to how humans perform, which not only restricts what we learn about model performance but also makes it difficult to get a sense of overall model behavior when these views are combined , let alone the similarities and differences between models, For these reasons, we have been interested in developing Global Model Analysis (GMA) techniques. They are intended to complement LMA, not replace it. The idea is that by stepping back from a particular data sample and obtaining a broader view of how a model performs, we can gain a deeper understanding of model behavior and how it compares with competing models.

One GMA measure is a model's complexity, which quantifies its inherent data-fitting ability in an experimental setting (Grunwald et al, in press; Pitt, Myung, & Zhang, 2002; Rissanen, 1996; 2001). The method is mathematically rigorous, as will be elaborated later, but a few limitations restrict its application to the diverse range of models in psychology. One is that it is currently only applicable to statistical (parametric, algebraically formulated) models. The other is that it generates only a single value. Although a useful starting point for exploration, by itself the measure cannot provide enough information about model behavior to meet our needs.

Recently, Navarro et al (in press; Navarro et al, 2003; Pitt & Navarro, in press; see also Wagenmakers et al, 2004) developed a GMA method called *Landscaping* that attempts to fulfill this goal while still maintaining ties with the complexity measure. A landscape is a plot of the relative fits of two models to a range of data sets generated by those models in a particular experimental design. Two examples are shown in Figure 1. The diagonal line represents the location of equal fit by the two models, with the distribution of points above the line indicating better fits by model A. The reverse is true of points below the line. The high degree of overlap of the distributions in the left panel indicates the models fit all data sets equally well. The separation of the distributions in the plot on the right indicates the models could be discriminable, most decisively if data were generated in the regions furthest from the criterion line, where one model would provide a far superior fit than the other.

Landscapes are relatively easy to create. In addition to learning about the types of data that can distinguish models, data collected in past studies can be overplotted on a landscape to assess their informativeness in distinguishing models. In short, the landscape provides a broader context in which to understand the relationship between two models and their fits to data.

Landscaping has two drawbacks, however. Like the complexity measure mentioned above, it too is restricted to statistical models. More importantly, comparison of fits cannot tell us much about the functional relationship between models. Are they virtually isomorphic in a particular experimental setup, generating the same number and range of data patterns, or are they distinct, with only a small subset of overlapping patterns between them? Goodness-of-fit measures might be relatively insensitive to these two situations. Even if such differences were detectable by fit quality, the reason for the differences could not be understood or taken advantage of without dissecting them further.

Exactly how to perform such analyses to address these issues is a nontrivial problem precisely because most models are complex, with many parameters that combine nonlinearly. We introduce a general-purpose version of landscaping that overcomes these two remaining hurdles and achieves this goal. It is called *Parameter Space Partitioning* (PSP), and involves doing exactly what the name implies: A model's parameter space is literally partitioned into regions that correspond to the data patterns that could be generated in an experiment. These partitions can be studied to learn about a model's behavior and compared across models to assess their similarities.

Parameter space partitioning: Peering into the black box of model behavior

To illustrate PSP approach, consider a visual word recognition experiment in which participants are asked to categorize stimuli as words or nonwords and response time to words is measured as the dependent variable across three experimental conditions, A, B and C. Suppose we are interested in the ordinal relationship (fastest to slowest) across conditions. In this case, there are 13 possible orderings (including equalities) that can be observed across the three conditions (e.g., A > B > C, A > B = C, B > C = A, etc). Each of these orderings defines a data pattern. Suppose that mean participant performance yielded the pattern B > C > A.

Now consider two hypothetical models, M_1 and M_2, of word recognition, each with two parameters. Using PSP, we can answer the following questions about the relationship between the models and the human data generated in the experiment: How many of the thirteen data patterns each model can produce? What part of the parameter space includes the human pattern, how much of the space is occupied by the human pattern? What data patterns are found in the rest of the parameter space?

Figure 2 shows the parameter space of each model partitioned into the data patterns it can generate. Model M_1 produces three, one of which is the human pattern. Note how the human pattern is central to the model, occupying the largest portion of the parameter space. Even though the model generates two other patterns, they differ minimally from the human pattern. In contrast, M_2 can produce nine of the thirteen patterns. Although one is the human pattern, M_2's performance is not impressive because it can mimic almost any pattern that can be observed in the experimental design; its predictive power is too great. Indeed, that M_2 mimics human performance seems almost incidental because the region of the parameter space that corresponds to the human pattern is small and some of the larger regions are produced by patterns that are unlike human data (e.g., C>A>B).

Challenges in Implementing PSP

The preceding example illustrates the gist of the method and describes some of what can be learned with it. Implementation of the method requires solving two non-trivial problems. One is how to define a data pattern, and the other is how to devise an efficient search algorithm to find the data patterns. Our solutions are described next.

Each set of a model's parameter values generates a model output, but not every model output is a distinct data pattern. How many qualitatively "different" outputs can a model produce?

Answering this question, which is what PSP enables us to do, depends critically on how a data pattern is defined. This is something which will vary from experiment to experiment, and indeed may vary within an experiment depending upon what a researcher wants to learn. Although there is no general solution, the experimentalist usually has a very good idea about what is going on in their data and what patterns should be found in order to confirm or disconfirm a model. Most of the time, one is testing ordinal predictions. In this case, a "natural" definition of a data pattern is the order relationship of model outputs, as in the above example. Nevertheless, it is a good idea to try out a couple of different definitions and to perform sensitivity analyses to ascertain if and to what extent conclusions obtained under one definition hold across others. An example of this is presented in the comparison of TRACE and Merge.

Once a data pattern is defined, the next challenge is to find all patterns a model can simulate. The data space by definition is made up of all patterns that can be observed in an experiment, however improbable. The space may contain a huge number of patterns, only a small fraction of which correspond to a model's predictions, so it is essential to use an efficient search algorithm that finds all of the patterns the model can generate in a reasonable amount of time.

The PSP Algorithm

The search problem to be solved is to partition a multi-dimensional parameter space into an unknown number of regions, each of which by definition corresponds to a unique data pattern. Given the dimensionality of the parameter space, brute force methods such as Simple Monte Carlo (SMC; a random search procedure) will not work. For each parameter set, we run the model, see what pattern it produces and keep track of only those that are unique. However, SMC is inefficient precisely because the search is random, not guided. Markov Chain Monte Carlo (MCMC; Gilk et al, 1996) is a much more powerful sampling method that we encorporated into an algorithm that efficiently finds all of the regions.

Application of the algorithm begins with a set of parameter values at which the model can generate a valid data pattern. This initial set can be supplied by the modeler or from an exploratory run using SMC. Given the parameter set and the corresponding data pattern generated by the model, the algorithm samples nearby points in the parameter space to map the region that defines the data pattern. MCMC is used to approximate quickly and accurately the shape of this region.

Figure 3 illustrates how the algorithm works in the space of a two-parameter model. The process begins with the initial parameter set serving as the current point in the parameter space (filled point in panel a). A *candidate* sample point (open circle) is drawn from a small, predefined region, called a jumping distribution, centered at the current point. The model is then run with the candidate parameter values and its output is evaluated to determine if the data pattern is the same as that generated by the initial point. If so, the candidate point is accepted as the next point from which another candidate point is drawn. If the new candidate point does not yield the same data pattern as the initial one, it is rejected as belonging to the current region. Another jump from the initial point is attempted, accepting those points that yield the same data pattern. The sequence of all accepted points recorded across all trials is called the *Markov chain*

corresponding to the current data pattern. This sample of points is used to estimate the size of the region occupied by the data pattern (panel b). The theory of MCMC guarantees that the sample of accepted points will eventually be distributed uniformly over the region. This feature of MCMC allows us to estimate accurately the volume occupied by the region, regardless of its size.

Every rejected point, which must be outside the current region, is checked to see if it generates a new valid data pattern. If so, a new Markov chain corresponding to the newly discovered pattern is started to define this new region. In effect, accepted points are used to shift the jumping distribution around inside the current region to map it completely, whereas rejected ones are used to discover new regions. Over time, as many search processes as there are unique data patterns (probable or not) will be run. Additional details about the algorithm are described in Appendix ???. [Insert fuller description as well as the Table detailing the steps in the Appendix]

Algorithm Evaluation

We tested the accuracy of the algorithm by measuring its ability to find all of the data patterns in a model that was simple enough for us to specify and manipulate their frequency and structure. The efficiency of the PSP algorithm was measured by comparing it to SMC (random search).

The model was a hypercube whose dimensionality *d* (i.e., number of parameters) was 5, 10, or 15.To illustrate the evaluation method, a two-dimensional model (d = 2) is depicted in Figure 4 that contains twenty data regions (outlined in bold) that the algorithm had to find. Note that a large portion of the space does not produce any valid data patterns. Also note that the sizes of the data regions vary a great deal. Some are elicited by a wide range of parameter values whereas others can be produced only by a small ranges of values. This contrast grows exponentially as the dimensionality of the model increases, and was purposefully introduced into the test to make the search difficult and approximate the complexities of nonlinear models.

Figure 5 shows performance of the PSP algorithm for a search problem in which there were one hundred regions embedded in a 10-dimensional hypercube (d = 10). The two curves are based on the average of 10 independent replications [Woojae: Figure should include standard deviations every 30 seconds.] The PSP algorithm found all the regions and did so in nine minutes. SMC found only about 23 patterns in nine minutes, and given its sluggish performance, it seems doubtful that SMC would find all of them in a reasonable amount of time.

Table 1 summarizes results from a series of tests comparing the two search processes, PSP and SMC. The dimensionality of the model was extend to 15 to make it comparable to models found in cognitive science. The number of data patterns that had to be found was deliberately made large (20 - 500) to make the test of the algorithm comprehensive and challenging. In addition, ten independent runs of each search method were carried out to assess the reliability of algorithm performance.

The mean proportion of patterns found is listed in each cell. The results are clear and consistent. The PSP algorithm almost always found all of the patterns whereas SMC failed to do so in every

condition. Most noteworthy is the success of the PSP algorithm (and its contrast with SMC) when there were many parameters and data patterns (lower right corner). The near perfect success of the PSP algorithm suggests it is likely to perform admirably in other testing situations. In the remainder of this paper we describe its applications to models in the fields of speech perception and category learning.

Comparing localist connectionist models of phoneme perception

The PSP algorithm was used first to compare two localist connectionist models. Connectionist models were chosen to demonstrate the method's ability in making inroads into understanding the behavior of models that exhibit a high degree of interconnectedness among their parts. The localist variety was chosen because of its popularity in some content areas (e.g., language; Grainger & Jacobs, 1998) and because we had previously worked with them (Brunsman, Myung, & Pitt, 1999).

TRACE (McClelland & Elman, 1986) and Merge (Norris, McQueen, & Cutler, 2001), two models of phoneme perception, were compared. Although similar in many ways, they differ with respect to how prior knowledge is combined with sensory input to yield a phonemic percept. In TRACE, word (prior) knowledge can directly influence phoneme processing. In Merge, such direct top-down feedback is not allowed. Rather, the two sources of information (word and phonemic) are integrated at a separate decision stage. A goal of this comparison was to assess the consequences of this design difference.

Schematic diagrams of the fundamental design properties of each model are shown in Figure ???. Note that both have a phonemic input stage and a word stage. Where they differ structurally is that the phonemic input stage also doubles as a decision stage in TRACE, but these functions are split between stages in Merge. The models also differ in the types of connections between stages. In TRACE there is bi-directional excitatory activation, as indicated by the arrows on both ends of the lines connecting phoneme nodes with word nodes. Within each stage, nodes inhibit each other. Merge, in contrast, has only bottom-up excitatory connections from the phoneme input stage to the word and phoneme decision stages. Inhibition is found only at the word and phoneme decision stages. It is absent at the phoneme input stage.

To compare the two models, it was necessary to equate them in every way possible to ensure that differences in performance were attributable only to design differences, not other factors, such as the size of the lexicon. In essence, we wanted to compare the fundamental structural and functional properties that define the models, nothing else. We did this by first implementing the version of Merge described in Norris et al (2001), and then making the necessary changes to Merge to turn it into TRACE (Norris et al essentially did this as well). When finished, the source code for the two models, written in Matlab, was identical except for the sections that corresponded to their design differences. TRACE required ??? fewer parameters than Merge (??? vs. ???) because the phoneme input and decision stages were combined. The names of the parameters, along with other model details, are in Appendix ???.

Norris et al (2001) proposed Merge as an alternative to TRACE because they felt the evidence

from the experimental literature did not warrant such a strong conclusion as direct word-to-phoneme feedback. Integration of phoneme and word information at a post-input decision stage is a more cautious way to combine lexical and phonemic information and, in their view, more in line with the data. The sufficiency of the Merge architecture was demonstrated in simulation tests in which it performed just as well as, if not slightly better than, TRACE in reproducing key experimental findings of how lexical knowledge affects phoneme processing.

We revisited two of these experiments (Frauenfelder, Segui, & Dijkstra, 1990; Marslen-Wilson & Warren, 1994) using the PSP algorithm to gain a global perspective of their behaviors. The models represent different theoretical views, but are they functionally different? Analyses were carried out on the PSP data to answer this and related questions.

Sensitivity to subphonemic mismatch

The first comparison of the two models was performed in the context of the subcategorical mismatch experiment by Marslen-Wilson and Warren (1994, Experiment 1; McQueen et al, 1999, Experiment 3). This experiment was chosen because of its complex design and the variety of response alternatives, which together permitted detailed analyses of model behavior at both the phonemic and lexical levels.

In the experiment, listeners heard one-syllable utterances and then had to classify them as words or nonwords (lexical decision task) or categorize the final phoneme (phonetic decision task). The stimuli were made by appending a phoneme (e.g., /b/) that was excised from the end of a word (e.g., *job*) or nonword (e.g., *smob*) to three preceding contexts, to yield six experimental conditions (listed in Table ???). The first context was a new token of those same items but with the final consonant removed (e.g., *jo* and *smo*), to create cross-spliced versions of *job* and *smob*. The second consisted of equivalent stretches of speech from two other words that differed only in the final consonant (e.g,, *jo* from *jog* and *smo* from *smog*). The third was the same as the second except that the initial parts were excised from nonwords (e.g., *jo* from *jod* and *smo* from *smod*).

Because cues to phoneme identity overlap in time (due to coarticulation), a consequence of cross-splicing is that cues to the identity of the final consonant will conflict when the first word ends in a consonant different from the second. For example, *jo* from *jog* contains cues to /g/ at the end of the vowel, which will be present in the resulting stimulus when combined with the /b/ from *job* (W2W1 condition).

Marslen-Wilson and Warren (1994; McQueen et al, 1999) were interested in how these affected phoneme and lexical processing. They found that in the lexical decision task, reaction times slowed when listeners heard cross-spliced stimuli, but responding was not affected by these conflicting cues (responses in the W2W1 and N3W1 conditions were equivalent). Phoneme categorization, in contrast, was sensitive to this subtle variation in phonetic detail, but only when the stimulus itself formed a nonword (e.g., *smob*).

Merge can simulate this complex pattern of results. TRACE can as well if the number of

processing cycles is increased from one to 15 (Norris et al, 2000). It is remarkable that both models can produce these data when one considers what is required of them. Not only does the correct word or phoneme node have to reach the activation (recognition) threshold before others, but it must do so at the appropriate rate across all conditions so that the profile of "recognition cycle times" matches that of human reaction times.

To determine the extent of their behavioral similarities, we analyzed the data generated from partitioning each model's parameter space to learn how many other data patterns the models can produce and how similar these patterns are across models and to the human data. The results yield a landscape of the relationship between the models and the experimental data that builds on the one in Figure 1 by providing information about patterns produced by the models.

Details of PSP analysis

Define data pattern. Mention how many there are.
Explain purpose of and define use of weak and strong constraints.
State how long (in hours) the searches took.

Results and Discussion

We began by analyzing classification phoneme and lexical performance, followed by analysis of the RT patterns.

Must explain the purpose of each analysis. Why is it being performed? What will it tell us? This is especially important because it is the first time they are introduced.

complexity - pattern count
volume ratios (*shared* vs unique; magnitude of human pattern)
volume correlations (shared only; ranking of human pattern, size+frequency of other patterns)
mismatch (error) distributions (similarity of competing patterns with human pattern)
mismatch vs volume correlations
micro analysis of mismatches (types of errors) - consistency analysis of errors.
RT analysis (consistency of ordinal predictions between conditions)
Parameter analyses

Indirect lexical inhibition and the bottom-up priority rule

The interactive architecture of TRACE makes it possible for word nodes to inhibit phoneme nodes indirectly. A word node sends top-down excitatory feedback to its constituent phonemes, which should in turn inhibit all other phonemes (see Figure ???). Frauenfelder et al (1990) tested this prediction in an experiment in which listeners had to monitor for phonemes that occurred late in multisyllabic words and pseudowords. Of interest was whether reaction times (RT) to the target phoneme would increase when presented stimuli that should caused indirect phoneme inhibition. RTs were compared among three conditions. A word condition (e.g., *cabinet*, with *t* as the target phoneme) served as a baseline in which lexical and phonemic information should combine to yield fast RTs. A control pseudoword condition (e.g., *vabinel*, with *l* as the target phoneme) also served as an upper limit on responding: Because *vabinel* is not a word, there should be no top-down lexical facilitation or inhibition when responding to /l/; responses should be based on sensory input alone. In the third, inhibitory condition, listeners heard pseudowords like *cabinel*, with *l* as the target phoneme. A slowdown in RTs relative to the word condition is expected if there is in fact inhibition. This is because the first part of the stimulus, *cabine* will activate *cabinet*, whose activation should then feed back down and excite /t/ and subsequently inhibit /l/.

The predicted RT slowdown in the inhibition condition was small and not reliable, which argues against direct word-to-phoneme excitation in TRACE. However, those predictions were formulated from a qualitative assessment of the model's behavior, not actual TRACE stimulations, which could differ. Inhibition might in fact be weak, or depend on other conditions not met in the experiment (stimulus properties such as word length or phonetic similarity). Norris et al (2001, p.317-320) compared TRACE and Merge in their abilities to simulate the human data pattern. Although TRACE's performance did in fact depend on stimulus characteristics, inhibition was found with longer (five-phoneme) stimuli. Merge produced no such inhibition.

One reason for Merge's correct performance is due to a property of the model not visible in the diagrams in Figure ???. Merge adheres to a bottom-up priority rule which states that phonemic input must precede lexical input in activating phoneme decision nodes. In the inhibition condition of the Frauenfelder et al experiment, because *t* did not occur at the end of the pseudoword *cabinel*, the *t* phoneme decision node would not be activated, a prerequisite for indirect lexical inhibition. [Footnote: One might wonder whether the bottom-up priority rule contributed to differences in model behavior in the subcategorical mismatch analysis. The reason it did not is because cross-splicing left remnants of a competing phoneme in the stimulus, which permitted phoneme-level inhibition. In the present test, the stimulus specifies only one phoneme, so there is no opportunity for inhibition.]

A PSP analysis was performed on TRACE and Merge in the Frauenfelder et al (1990) experimental setup to develop a clearer understanding of their behavior in what has the potential to be a test that can distinguish the two models. Because model performance differences could be attributable to two properties of the models, either their structural differences or Merge's priority rule, we evaluated the influence of both.

Details of PSP implementation

Mention that we included a lexical level response in the simulations because humans perceive the items as a word or pwords, even though participants made no lexical decision response. Imposition of this requirement narrows the parameter space because of the lexical restriction. This additional constraint on model performance should increase the opportunity to find differences between them because the region containing human data will be smaller. (Without this constraint, do the model perform identically? i.e., same # of patterns?)

State how long (in hours) the search took. Discuss reliability of results if search were run again.