SPECIAL ISSUE

Articles

TOWARD MEMORY-BASED REASONING

The intensive use of memory to recall specific episodes *from the past—rather than* rules*—should be the foundation of machine reasoning.*

CRAIG STANFILL and DAVID WALTZ

The traditional assumption in artificial intelligence (AI) is that most expert knowledge is encoded in the form of rules. We consider the phenomenon of reasoning from memories of specific episodes, however, to be the foundation of an intelligent system, rather than an adjunct to some other reasoning method. This theory contrasts with much of the current work in similarity-based learning, which tacitly assumes that learning is equivalent to the automatic generation of rules, and differs from work on "explanation-based" and "case-based" reasoning in that it does not depend on having a strong domain model.

With the development of new parallel architectures, specifically the Connection Machine® system, the operations necessary to implement this approach to reasoning have become sufficiently fast to allow experimentation. This article describes MBRtalk, an experimental memory-based reasoning system that has been implemented on the Connection Machine, as well as the application of memory-based reasoning to other domains.

THE MEMORY-BASED REASONING HYPOTHESIS

Although we do not reject the necessity of other forms of reasoning, including those that are currently modeled by rules or by the induction of rules, we believe that the theory behind memory-based reasoning warrants further experimentation. Two bodies of evidence have led us to support this hypothesis. The first is the study of cognition: It is difficult to conceive of thought without memory. The second is the inability of AI to achieve success in any broad domain or to effectively capture the notion of "common sense," much of which, we believe, is based on essentially undigested memories of past experience.

We will primarily discuss the application of our hypothesis to the limited case of *similarity-based induction*. The goal of similarity-based induction is to make decisions by looking for patterns in data. This approach has been studied extensively in the context of "similarity-based learning," which uses observed patterns in the data to create classification rules. The memory-based version of this method eliminates the rules, solving the problem by direct reference to memory. For example, given a large set of medical records, a similarity-based induction program should be able to diagnose new patients (classify them as having some known disease), using only the information contained in the database. The two approaches differ in their use of the database: Rule induction operates by inferring rules that reflect regularities in the data, whereas memory-based reasoning works directly from the database.

Primitive Operations of Memory-Based Reasoning

The memory-based reasoning hypothesis has not been extensively studied in the past because von Neumann machines do not support it well. There have been systems, such as Samuel's Check-

The Connection Machine is a registered trademark of Thinking Machines Corporation.

© 1986 ACM 0001-0782/86/1200-1213 75¢

ers player [38, 39], which worked from memory, but they have required an *exact match* between an item in memory and the current situation, and a non-general search algorithm (e.g., hashing or indexing) to retrieve items from memory. In the real world there may be no exact matches, so the "best match" is called for. Unfortunately, there is no general way to search memory for the best match without examining every element of memory. On a von Neumann machine, this makes the use of large memories impractical: Experiments might take weeks to run. On a fine-grained parallel machine, such as the Connection Machine system, this problem does not begin to arise until the size of the database exceeds the size of the machine's memory, which pushes the problem back three or four orders of magnitude. (For another application of the Connection Machine system to a data-retrieval problem, see "Parallel Free-Text Search on the Connection Machine System" [43] on page 1229 of this issue.)

The currently available Connection Machine system [11] is a massively parallel SIMD computer with up to 65,536 (2^{16}) processing elements, each having 4,096 bits of memory and a 1-bit-wide ALU.[1] The architecture can (in principle) be scaled upward to an arbitrarily large number of processors. Every processor executes the same program, though processors can selectively ignore program steps. Processors are connected to each other via a hypercube-based routing network. On a Connection Machine system with n processors, any processor can send a message to any other processor in $\log n$ time.

Accessing memory, in our current paradigm, requires four basic operations. The first is counting the number of times various features or combinations of features occur. For example, in the context of medical applications, if we had a patient with a high fever, we would want to find out how often a high fever occurred in combination with various diseases. The second is using these feature counts to produce a metric (a topic discussed in detail in the section on the basic algorithm). The third step is calculating the dissimilarity between each item in memory and the current case (patient), and the last is retrieving the best matches. The first operation—counting—will be described here briefly as an illustration of how the parallelism of the Connection Machine system is utilized.

One term in our metrics is the probability of a particular "disease" A given some "symptom" B (the conditional probability $A \mid B$). To compute this, we count the number of data items having feature B, count the number of data items having both features A and B, and divide.[2] The counting operation can be done for every symptom–disease combination simultaneously, in $\log^2 n$ time. (For a discussion of the principles underlying such algorithms, see [12].)

The amount of hardware needed to build a Connection Machine system is $O(n \log n)$ in the number of processors.[3] The time needed to perform memory-based reasoning is $O(\log^2 n)$. Therefore, memory-based reasoning scales to arbitrarily large databases, as neither the quantity of hardware nor the time required for processing grows at a prohibitive rate.

Memory-Based Reasoning in More Detail

Although the basic idea of memory-based reasoning is simple, its implementation is somewhat more complex: We must separate important features from unimportant ones; moreover, what is important is usually context-sensitive. One cannot assign a single weight to each feature and expect it to hold true for all time. For example, suppose a patient reports abdominal pain, which happens to match both records of pregnant patients and patients with ulcers. In the case of ulcers, gender is irrelevant in judging the similarity of the record to that of the patient, whereas it is highly relevant in the case of pregnancy. However, this fact can only be known to the system if it computes the statistical correlations between gender and ulcers on one hand, and gender and pregnancies on the other. Thus, we cannot assign a single weight to gender ahead of time; instead, it must be calculated with respect to the task at hand.[4] Various schemes for accomplishing this will be presented below.

The operation of memory-based reasoning systems can be made clearer through another example. Suppose that we have a large relational database of medical patient records, with such *predictor features* as "age," "sex," "symptoms," and "test results for each patient." Memory-based reasoning operates by proposing plausible values for *goal features*, such as "diagnosis," "treatment," and "outcome," which are known for the patients in the database, but are unknown for the patient being treated. A program can fill in the "diagnosis" slot for a new patient by

[1] Machines with 2^{14} and 2^{15} processors are also available.

[2] This method would be sufficient to support Bayesian reasoning [2, 3, 21] in real time, working directly from a database.

[3] We are counting the number of wires in the hypercube. A hypercube with 2^k vertices (processors) will have $\frac{1}{2}k2^k$ wires. Setting $n = 2^k$, the number of wires is $\frac{1}{2}n \log_2 n$. The hardware needed to construct the processors is, of course, linear in the number of processors.

[4] One can imagine saving weights for various situations rather than always calculating them on the fly, but this puts heavy demands on memory and requires another matching process between the current situation and prestored weights for a record.

(1) broadcasting to each processor the values for each of the new patient's predictors; (2) having each processor compute a numerical "distance" and a numerical "weight" for each predictor in its record as a function of the new patient's value for that predictor;[5] (3) having each processor use these weights and distances to compute a total distance measure from each record to that of the new patient; and (4) selecting the n patients whose total distances from the new patient are the smallest.

There are several possible outcomes: (1) No patient is sufficiently similar to the current case to make a diagnosis; (2) a very small number of patients (one or two) are retrieved; (3) a significant number of patients are retrieved, and all have similar diagnoses; and (4) there are several diagnoses among the nearest n patients.

At this point, we can make some observations: If the first case holds, the system knows that it has never seen this set of symptoms before, and thus, it "knows that it doesn't know" how to diagnose this case. (Alternatively, the data on the new patient may be in error; the system can also propose this possibility.) In the second case, even if only one patient is similar (e.g., as in the second case of Legionnaire's disease), the system may be able to make a tentative diagnosis. If the third case is true and all the retrieved patients have the same diagnosis, it is very likely that this patient should also get the same diagnosis; we might even be able to extract a diagnostic rule for this condition, which could be added to a microcomputer-based expert system. Finally, if the fourth case is true, then the system knows that it cannot come up with a definitive answer, and has several options: It can acquire more information (e.g., order blood tests), it can choose the most likely diagnosis, or it can propose working with an uncertain diagnosis (e.g., if it thinks the patient might have pneumonia, but is not certain, it can suggest prescribing antibiotics as a precaution).

Thus, memory-based reasoning, as applied to similarity-based induction, seeks to make decisions by "remembering" similar circumstances in the past. This is done by counting combinations of features, using these counts to produce a metric, using the metric to find the dissimilarity between the current problem and every item in memory, and retrieving the best matches. When implemented on the Connection Machine system, this takes $\log^2 n$ time. Since neither the hardware nor the processing time required grows at a prohibitive rate, memory-based reasoning scales to arbitrarily large databases. Memory-based reasoning degrades gracefully when it cannot come up with a definitive answer to a problem: It may respond that no answer is possible, give one or more plausible answers, or ask for more information.

[5] This is the complicated part of the algorithm. In general, a particular diagnosis will correspond to a range of values for a predictor. If both the new patient's value and a particular record's value tend to suggest the same diagnosis, the "distance" measure between the two predictors should be small, whereas if the two values are different, and the new patient's value rarely if ever occurs with the diagnosis in the record, the distance between the two should be large. The "weight" measures how important the predictor is: If the particular predictor value is associated only with this diagnosis, then the weight should be high, whereas it should be low if the same predictor value is correlated with many different diagnoses. Consider body temperature as a predictor: 98.6 and 98.8 would have a small distance between them, but a very low weight in predicting, say, cancer; 103.0 and 104.0 have a fairly large distance between them, but also have quite a high weight for predicting diseases that are associated with high fevers.

BACKGROUND: AI PARADIGMS

Memory-based reasoning is related to a number of other subfields of AI, such as expert systems, machine learning, and "connectionist" research. It is also related to several fields outside of AI proper: statistics, cognitive science, information science, and database research. Here we will explore this work in the context of AI.

For 30 years heuristic search and deduction have been the dominant paradigms in the central research areas of AI, including expert systems, natural-language processing, and knowledge representation. This paradigm was applied to a wide range of problems, and met with many early successes, including the General Problem Solver [32], the Geometry Theorem Prover [9], the MYCIN expert system for medical diagnosis [42], resolution theorem proving [35], and parsing with Augmented Transition Networks [47]. Each of these programs depended on searching for the right set of operators to reach a goal state. Researchers have progressed by devising methods for specifying the operators, refining the structure of the information they operate on, and improving control over which operators are applied in each situation. Expert systems are solidly within this paradigm: In an expert system, operators are generally called *rules*, and the goal state is a *decision* (e.g., a diagnosis in a medical expert system) or *action* (as in an expert system designed to control a process). The AI languages Planner [10], Micro-Planner [44], and Prolog [18] all embody some variant of heuristic search with backtracking as their central control mechanism.

The reasons why this paradigm has dominated the field are not difficult to understand in retrospect. AI originated in the early 1950s and was strongly influenced by work on the analysis of human problem-solving protocols in psychology [26] and AI [31, 33]. Humans experience problem solving as trial and error: Much effort seems to be spent in deciding

which operations to apply and in recovering from the consequences of wrong choices. At least for fairly small, well-defined problems, this model proved a good match to the abilities of von Neumann computers, which could perform small, well-defined operations, one at a time.

Problems with the Heuristic Search Paradigm

Although it is clear that intelligent behavior often resembles heuristic search, at least at a gross level, there is no reason to suppose that heuristic search plays much of any role in the processes that underlie the operations of consciousness. As pointed out frequently by such critics of AI as Dreyfus [5], it is the underlying processes that account for the vast range of human behavior: Only a few kinds of planning and problem-solving tasks are accounted for by heuristic search methods, while in day-to-day life a vast range of our activities is handled by very quick decision-making processes that involve no exploration of alternatives—where the reasons we give for taking these actions are often justifications generated in retrospect. Heuristic search cannot neatly explain the reasons for actions that we attribute to intuition, nor can it explain the wide range of actions that we can perform within 200 milliseconds, which include such problem-solving activities as goal and operator selection, in addition to everyday tasks such as reacting to complex situations while driving, using tools and utensils, and answering questions about facts or experiences.[6]

Moreover, practical applications of the heuristic search/deductive reasoning paradigm have had only limited success to date. (See [45] for an extended discussion of the problems and state of the art in expert systems.) *Knowledge acquisition*, or the process of identifying and formalizing rules, remains an art. No truly automatic knowledge-acquisition scheme has been devised; instead, *knowledge engineers* must collaborate with expert informants[7] in the domain of application, training them to think about their decisions and actions in terms that can be turned into rules and knowledge bases. This process often takes a very long time (a year or more). In the end the performance of such systems has often been disappointing. Rule-based expert systems are "brittle"—they tend to fail badly for problems even slightly outside their area of expertise and in unforeseen situations. The systems are not capable of detecting patently absurd conclusions, unless rules are explicitly inserted to detect them.

[6] As pointed out by Feldman [7], in order to act or decide within 200 milliseconds, the total length of the longest path between input sensors and motor effectors can involve no more than about 100 neurons. This seems to preclude neural backtracking.

[7] Except, of course, where the knowledge engineer is also an expert in the domain of application.

Mainstream Learning in AI

Because the scope of AI systems is limited until they can learn to extend themselves from experience, learning has always been a major area of research in AI.[8] Most of this work falls into two "mainstream" camps: similarity-based and explanation-based learning. Similarity-based learning has generally been expressed as the induction of either rules or class definitions [20, 24, 28, 34] from examples. The archetype for this paradigm was Winston's ARCH program [46], which inferred the definition of an arch from a series of examples. Of special relevance to our work is Lebowitz's system for discovering categories in a database combining voting districts' demographics and legislative voting records [19]. Cheeseman has proposed a variation on similarity-based reasoning that creates a statistical model of a database, then uses this model to make decisions [3]. He claims that this decision procedure can be proved optimal.

Similarity-based learning and memory-based reasoning are analogous in that they solve the same problem: using syntactic patterns in a database to solve classification and decision problems. They differ substantially in mechanism, in that similarity-based learning uses rules as an intermediate structure, whereas memory-based reasoning operates directly on the data.

The other primary school of learning research is the explanation-based learning paradigm [4]. This is an incremental, knowledge-based approach, in which a system learns by explaining unanticipated events. The stated goal is "one-shot learning," in which learning can proceed from a single example. The classical example is learning the description of a cup: A program is given a structural description of a cup, the purpose of a cup (it allows one to drink), and domain knowledge sufficient to find the connection between structure and function. The program discovers that the handle allows the cup to be lifted, that the base prevents the cup from tipping over, and that the bowl allows it to contain liquid. Other aspects of the cup's structure, such as its color, composition, and precise shape, are not used in constructing the explanation and are hence left out of the general description of a cup. (See [27] and [29] for recent treatments of this problem.)

Explanation-based learning presupposes strong domain models, which are conventionally implemented through rules or through data structures traversed by a fairly uniform deductive procedure. Ultimately, we will want to incorporate explanation-based techniques into a general theory

[8] References [24] and [25] offer broad collections of articles on current research issues in learning.

of memory-based reasoning. To do so now, however, would be premature, as we are working at a level where such domain models are unavailable.

Current work on *case-based reasoning* [1, 17, 40] is also relevant, as it combines memory with explanation-based learning to allow a program to learn by remembering and analyzing its mistakes. In this model, when faced with a difficult task a problem solver will search memory for a previous case where an analogous task arose, then try to adapt that solution to its current needs. Like explanation-based learning, it requires a strong domain model.

Other Paradigms for Learning

Since the late 1970s a great deal of increased attention has been given to systems that learn using entities other than symbolic ones. This work harks back more to the perceptrons and pattern recognition of the 1950s and early 1960s [36] than to the "physical symbol system" models of mainstream AI [30]. Some research, for example, Holland's "genetic algorithms" systems [14], has persisted since the early 1960s; most (e.g., [7, 13]) had its origin much more recently. The extensive literature on "associative memory" [13, 16] is also relevant, but these systems are generally oriented more toward deduction than induction.

The bulk of the work in nonsymbolic learning has taken place in the field loosely called *connectionism.* Connectionist systems involve computations using large networks of linked nodes where weights on links and nodes are the loci of learning and computation. There are two broad interests in connectionist systems: representation [6, 8] and learning [13, 15]. (See also [22] for a comprehensive treatment.) In general, nodes are assigned numerical "activation levels," and a means of computing outputs from the sum of the input link activations (e.g., thresholds or logistic functions). Links are directed and have weights; node outputs are multiplied by each link's weight to compute that link's influence on the node it ends on. Representation concerns the mapping between concepts and nodes, and is outside the scope of this article.

Learning in a connectionist network involves adjusting the link weights or node thresholds to new values so that the computations of the network will more closely match desired performance. Although there are a variety of ways for a connectionist network to learn, we will only consider here a method called *back propagation error learning* [37]. In this method, an input is presented to a system that has random link and node weights. The output generated for this input is compared with the desired output, and an error signal is computed for each of the output units. This output error is then propagated back to the links that feed the output units, then accumulated at internal *hidden units* (which are neither input nor output units). This error is then propagated again until it reaches the input units. As errors propagate past links, their weights are adjusted so as to lessen errors. This method, a kind of stage-by-stage steepest descent, seems to provide near-optimal solutions, and to provide them faster and in a conceptually clearer manner than in earlier connectionist models.

NETtalk [41] was one of the first systems to use back propagation methods. NETtalk performs a word pronunciation task similar to the one solved by MBRtalk, described below, the difference being that it operates in the connectionist paradigm, whereas MBRtalk operates in the memory-based reasoning paradigm. A series of words are presented to NETtalk, and its output for each is compared to the "correct" pronunciation from *Merriam-Webster's Pocket Dictionary* [23]. NETtalk uses the results of this comparison to adjust its internal weights, so that its performance gradually improves.

Internally, NETtalk consists of a three-layer connectionist network. The three layers are a set of input units, a set of hidden units, and a set of output units. Each of the input units corresponds to a position of a letter in a word. There are seven sets of input units; in general, NETtalk's task is to provide a pronunciation for the letter in the center set of units based on its letter and the three letters that precede and the three letters that follow it. Each of the groups of seven input nodes consists of nodes for each of the letters of the alphabet, space, and so on. Each of the output units corresponds to a phonetic feature. In the task, input words are presented one at a time and "slid across" a window of input units so each of their letters appears in succession in the center set of units. Errors are accumulated at each of the output units, and back propagated after each letter is processed.

After 30,000 trials, NETtalk generated the correct phoneme 94 percent of the time for words it had already been "taught"; it generated the correct phonemes 78 percent of the time on novel words. The actual number of fully correct *words* is, of course, substantially less.

PRONOUNCING ENGLISH WORDS

The first task we discuss here is that of pronouncing novel words, using a database generated from 4438 English words and their pronunciations. The program that performs this task is called MBRtalk. Each of the 4438 words is used to generate as many database records as it has letters. Each of these records

consists of a letter, the preceding 4 letters, the succeeding 10 letters, the phoneme and stress for the letter, and the preceding 4 phonemes and stresses.[9] The program then tries to infer the pronunciation of words not in the database.

Discussion of the Task

MBRtalk achieves an 86 percent correct *phoneme* rate on novel words; however, this figure can be somewhat misleading, so we have done most of our calculations on the percentages of *words* done correctly. MBRtalk yields a full word performance of 47 percent correct, 21 percent marginally wrong, and 32 percent wrong.

The 94 percent correct phoneme rate of NETtalk was achieved by testing with the same set of words on which it was trained. For MBRtalk this kind of test makes no sense, since an input word that exactly matched a word stored in the system would have zero distance and would always dominate the decision making. Consequently, testing on the words in the database is too easy for our system (though an interesting and meaningful test on NETtalk). For novel words, the performance of NETtalk and MBRtalk seems comparable.

NETtalk and MBRtalk both take advantage of regularities in pronunciation, but no system can ever achieve perfect performance on this task. For example, there are irregular words, such as *though* versus *tough*. If such irregulars are omitted from the training set, then the system has no way of reliably pronouncing them. If, on the other hand, the irregulars are part of the training set, they create noise that might prevent the system from seeing the correct pattern. Furthermore, English has borrowed many words from other languages, often retaining their original pronunciations. No automatic system could properly infer the pronunciations of fil**ET** versus targ**ET**, v**ILLA** versus tort**ILLA**, pi**ZZ**a versus fi**ZZ**y, unless it had also been programmed to recognize the likely language of origin. Thus, the pronunciation task is an extremely challenging one for any inductive algorithm, as the data are noisy, the program is working with incomplete information, and such rules as do exist always have exceptions.

MBRtalk and NETtalk start with the same set of words and pronunciations. A pronunciation key is shown in Appendix A.

THE BASIC ALGORITHM

In this section we will present the current algorithm for memory-based reasoning. Our goal is not so much to produce the best possible algorithm as to produce one that is adequate to test the memory-based reasoning hypothesis.[10]

The memory-based reasoning hypothesis is that reasoning may be accomplished by searching a database of worked problems for the "best match" to the problem at hand. This requires a means of judging how closely two situations match, leading to the topic of metrics. A metric is a distance measure Δ satisfying the following four properties:

$$\Delta(a, b) \geq 0$$

$$\Delta(a, b) = \Delta(b, a)$$

$$\Delta(a, a) = 0$$

$$\Delta(a, b) + \Delta(b, c) \geq \Delta(a, c)$$

The implementation of memory-based reasoning depends on finding a suitable definition of Δ.

Notation

Before we can discuss metrics, we must define some terms. A *record* is a structured object containing a fixed set of *fields*. We use Greek letters (τ, ρ) for records, and italics for field names (n, p). A field may be empty, or it may contain a value. We use quoted, "typewritten" letters ('a', 'b') for specific values, the letter u for nonspecific values, and λ for empty fields. Field f of a record ρ is written $\rho.f$. The set of possible values for a field f is written V_f. A nonspecific value is represented by an italic letter (v_1). A *database* D is a set of records.

A *target* is a record τ containing some empty fields. The empty fields are called *goals*, and the nonempty fields are called *predictors*. The set of goals is written G_τ, and the set of predictors is written P_τ.

A *feature* is a combination of a field and a value, such as $[f = v]$. We use features to restrict a database to a subset, as in $D[f = v]$. We can count the number of items in the full database, as $|D|$, or in a restricted database, as in $|D[f = v]|$.

Representations for Words

For the pronunciation task, we have one record for each letter in a word. Each record has nine fields: the previous three letters (field $n - 3$, $n - 2$, and $n - 1$), the letter itself (field n), the next three letters ($n + 1$, $n + 2$, and $n + 3$), the phoneme assigned to that letter (field p), and the stress (field s).[11] The seven letter fields ($n - 3, \ldots, n + 3$) are predictors, while the phoneme and the stress fields (p and s)

[9] To simplify the discussion, we sometimes simplify the record to include only the preceding three letters, the succeeding three letters, the phoneme, and the stress. This is the representation used by NETtalk and in preliminary versions of MBRtalk. The reasons for the expanded record format are explained in the next section.

[10] Readers well versed in mathematical statistics, decision theory, or automatic pattern recognition may object that this algorithm is ad hoc in the sense that we cannot offer any rigorous explanation of why it works. The point is well taken, and more rigorous examination will probably produce a better algorithm, but nevertheless this algorithm produces the right answer often enough to support the memory-based reasoning hypothesis.

[11] As noted in Appendix A, this is a simplification of the actual representation.

are goals. The possible values for the letter fields ($V_{n-3}, \ldots, V_{n+3}$) are the letters of the roman alphabet plus the blank (denoted ·). The possible values of the phoneme field (V_p) are the phoneme codes noted in the previous section, plus the silent (–). The possible values of the stress field (V_s) are '1' (primary stress), '2' (secondary stress), '0' (unstressed), '+' (rising), and '–' (falling). For the world *file* we get the following four records:

A shorter version is written as follows:

```
◇ ···file f + ◇
◇ ··file· A 1 ◇
◇ ·file·· l – ◇
◇ file··· – – ◇
```

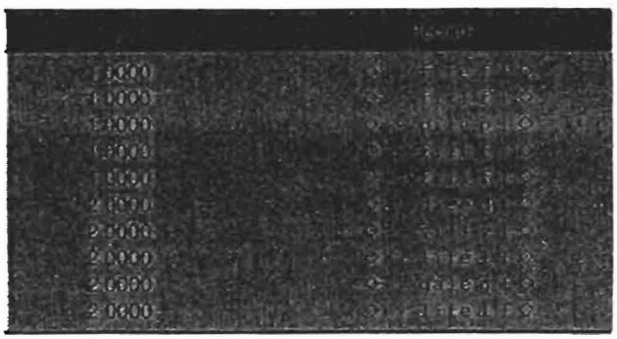
The Overlap Metric

The simplest measure of dissimilarity between two records τ and ρ is the number of fields f for which they have different values. We call this the *overlap metric*. It is not a particularly good metric, but makes a convenient starting point. We define it as follows:

$$\Delta(\tau, \rho) = \sum_{f \in P_\tau} \delta(\tau.f, \rho.f)$$

$$\delta(\tau.f, \rho.f) = \begin{cases} \tau.f = \rho.f & 0 \\ \text{otherwise} & 1 \end{cases}$$

This metric is not very useful because it assigns equal weights to all fields. To illustrate this, we took the record ◇ ···file λ λ ◇ and applied the above metric to a database of 8192 records. The following are the 10 best matches:



Obviously, all fields should not be equal in importance.

The Weighted Feature Metric

Features differ in importance because they differ in the strength with which they constrain the values of goal fields. For example, the feature [n = 'f'] constrains the field n to be 'f'. Among records satisfying this constraint, the only observed values of the field p will be the phonemes 'f' and '–' (silent). Thus, [n = 'f'] is very useful in predicting $\tau.p$ and should be given a high weight. On the other hand, the feature [$n + 3$ = 'e'] is not a good predictor of $\tau.p$.

Different values of a single predictor field can also differ in how strongly they constrain the values of a goal field. For example, the feature [n = 'a'] constrains $\tau.p$ to be one of a relatively small number of phonemes, such as 'a' (bAr), 'e' (bAy), 'c' (All), or 'x' (About). Other values of $\tau.n$ impose stronger constraints on $\tau.p$; as noted above, [n = 'f'] constrains $\tau.p$ to be either 'f' or '–'.

We incorporate this into our metric by giving different weights to different features, starting with a predictor field f, a goal field g, a target τ, and a database D. We determine the feature's weight $w_f^g(D, \tau.f)$ by measuring the strength of the constraint imposed by the feature [$f = \tau.f$] on the field g. The specific method is to restrict the database to $D[f = \tau.f]$, find the frequencies of the various values of g, square them, sum them, and take the square root of the result. The resulting metric is shown below:

$$\Delta^g(D, \tau, \rho) = \sum_{f \in P_\tau} \delta_f^g(D, \tau.f, \rho.f)$$

$$\delta_f^g(D, \tau.f, \rho.f) = \begin{cases} \tau.f = \rho.f & 0 \\ \text{otherwise} & w_f^g(D, \tau.f) \end{cases}$$

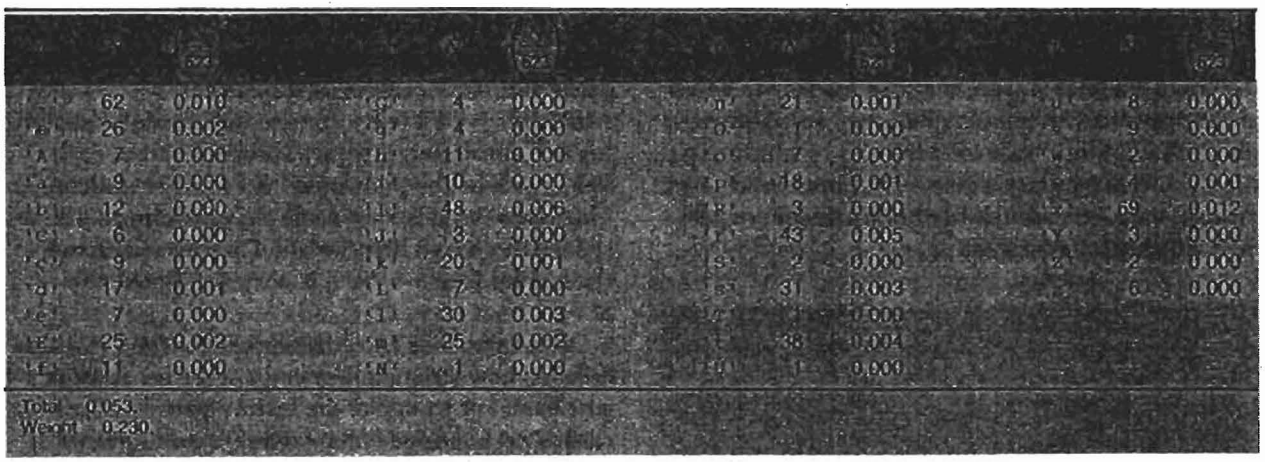
$$w_f^g(D, \tau.f) = \sqrt{\sum_{v \in V_g} \left(\frac{|D[f = \tau.f][g = v]|}{|D[f = \tau.f]|} \right)^2}$$

Let us calculate w_n^p and w_{n+3}^p for the record ◇ ···file λ λ ◇. We start with a database of 8192 records. The first step is to restrict the database to [n = 'f'], leaving 120 records. We then count the number of times each possible value v of the field p occurs and use the answer to compute the weight:

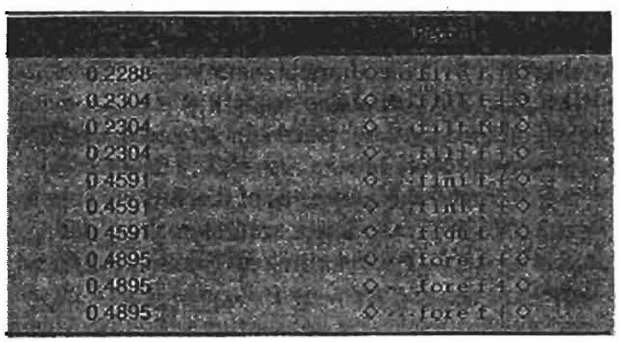
Total = 0.820.
Weight = 0.906.

We can also do this calculation for [$n + 3$ = 'e'] as shown in Table I (next page). This metric is ap-

TABLE I.



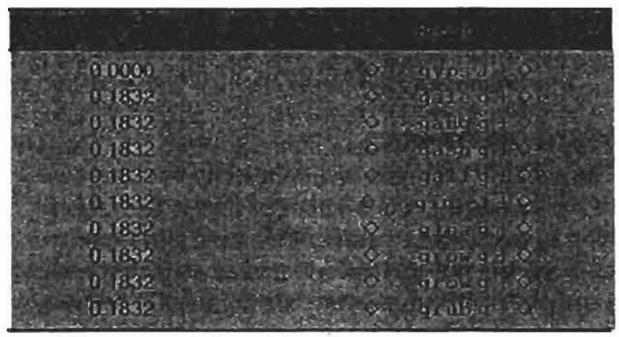
plied to a database of 8192 records. The 10 nearest matches to ◇ ···file λ λ ◇ are shown below. The improvement over the previous metric is clear:



Value Differences

The metric shown above is still too strict in that it is based on the precise equality of the values of predictor fields. In some cases different values of a predictor field will be equivalent. For example, the letter 'g' may yield four different phonemes: 'g' (as in Globe), 'J' (as in German), 'f' (as in rouGh), and '—' (as in althouGh). Let us consider 'g' and 'J'. We find that *gy*, *ge*, and *gi* are usually pronounced 'J' whereas all others are generally pronounced 'g'. Thus, for determining the pronunciation of 'g', the letters 'e', 'i', and 'y' are similar to each other, and different from all other letters.

Suppose, now, that we are trying to pronounce the word *gypsum*. Using the above metric, we get the one word in the database that contains 'gy' (gypsy), as well as a more or less random assortment of other words containing the letter 'g'. As most 'g''s (taken at random) are hard, the memory-based reasoning algorithm will conclude that the 'g' should probably be pronounced with the hard sound. The output shown below is the result of running the target ◇ ···gyps ◇ against the restricted database $D[n = \text{'g'}]$ (the reason for this restriction will be explained in the next section):



We can do better than this by modifying the metric to take into account similarity of values. What we need is a measure of the difference of two features, so that the penalty assessed a field will be the product of the field's weight times this difference measure.

Suppose we are given a target τ, a record ρ, and a database D. To compute the difference d_f^g between two predictive features $[f = \tau.f]$ and $[f = \rho.f]$, we calculate the frequencies of the various possible values of the goal field g in the restricted databases $D[f = \tau.f]$ and $D[f = \rho.f]$, subtract them, square the results, and sum over all values of g. This is the *value difference metric*:

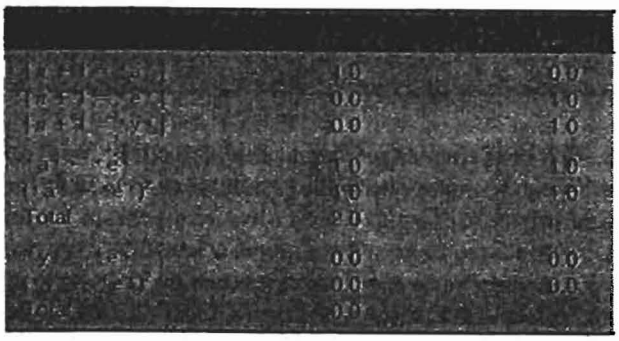
$$\Delta^g(D, \tau, \rho) = \sum_{f \in P_\rho} \delta_f^g(D, \tau.f, \rho.f)$$

$$\delta_f^g(D, \tau.f, \rho.f) = d_f^g(D, \tau.f, \rho.f) w_f^g(D, \tau.f)$$

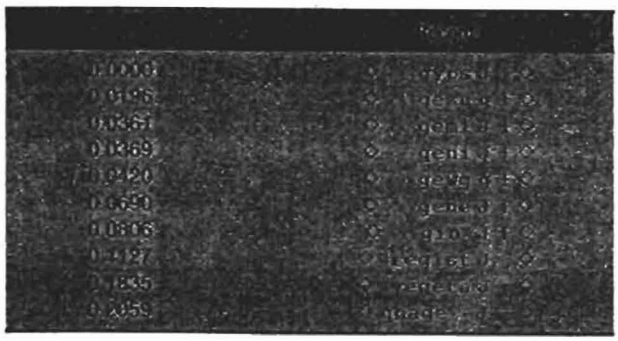
$$w_f^g(D, \tau.f) = \sqrt{\sum_{v \in V_g} \left(\frac{|D[f = \tau.f][g = v]|}{|D[f = \tau.f]|} \right)^2}$$

$$d_f^g(D, \tau.f, \rho.f) = \sum_{v \in V_g} \left(\frac{|D[f = \tau.f][g = v]|}{|D[f = \tau.f]|} - \frac{|D[f = \rho.f][g = v]|}{|D[f = \rho.f]|} \right)^2$$

For example, to calculate the difference between 'a' and 'e' in distinguishing between 'g' and 'J', we calculate the frequences of 'g' and 'J' for each, then subtract, then square. This yields a field difference of 2. For 'e' and 'y', the same calculation yields a difference of 0:



When we apply this metric to our original problem, we find more support for the notion that the 'g' in *gypsum* is to be pronounced 'J':



Restricting the Database

If predictors always acted independently in constraining the values of goals, the algorithm we have described would be sufficient. Such is not the case, as the combined effect of two predictors is often quite different than their effect separately. For example, the combination of the feature [n = 'g'] with [$n+1$ = 'a'] lets us predict that the phoneme p will be 'g' and not 'J'. However, if we look at the feature [$n+1$ = 'a'] in isolation, the effect vanishes.

The solution to this problem lies in a technique called *restriction*—applying the memory-based reasoning algorithm not to the entire database, but to a subset. We may obtain this subset by restricting the value of a predictor field or by restricting the value of the goal field. These two techniques are called *predictor restriction* and *goal restriction.*

Predictor restriction is accomplished by finding the most important field (as judged by the weight function w_f^g) and restricting the database to those records having the same value in that field as the target record. In the "gypsum" example shown above, we calculate the weight for each of the predictor fields and find that the feature n has the highest weight. The value of n in the record ◇ ···gyps λ λ ◇ is 'g', so we restrict the database to [$n = g$]. We then apply the field difference metric as described above.

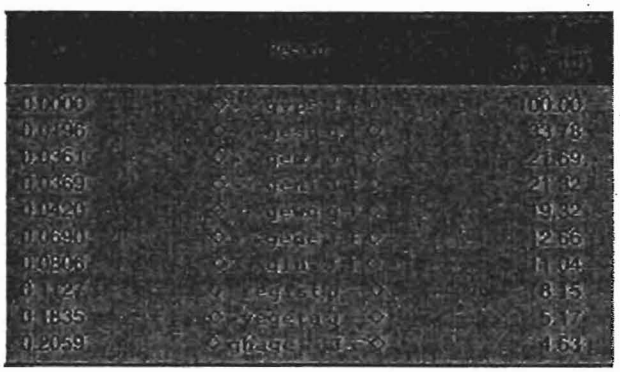
Goal restriction is accomplished by using the memory-based reasoning algorithm to discover plausible values for the goal field, then restricting the database to records containing one of those values in their goal fields. In our example of "gypsum," applying the field difference metric to the entire database yields 'g' and 'J' as plausible values for p. If we restrict the database records having either 'g' or 'J' as their phonemes and then apply the field difference metric to the reduced database, we get results similar to those shown for predictor restriction.

Summation of Evidence

Once we have found the dissimilarity between the target and every record in the database, we need to decide on the most likely value for the goal field.

The first step is to retrieve the records that most closely match the target. We might do this by setting a threshold and retrieving all records with smaller dissimilarity ratings. Alternatively, we could retrieve the n closest matches, for some suitable value of n. At present, we use the latter method, with $n = 10$.

Having retrieved the records most closely matching the target, we look at their goal fields. If the goals are all the same, we can be confident that the value they contain is the correct answer. Sometimes, however, several different values of the goal field will be observed. In this case we must decide which of these values is preferred. We assign each record a weight equal to the reciprocal of its dissimilarity[12] and sum the weight of each observed value of the goal field. The example below shows this weighting method:



[12] We first add 0.01 to avoid dividing by zero. We could also give a dissimilarity of zero infinite weight. The latter alternative is appropriate if the value of the goal field is a function of the values of the predictor fields. This condition does not hold for the pronunciation task as formulated here, as a word may have two different pronunciations.

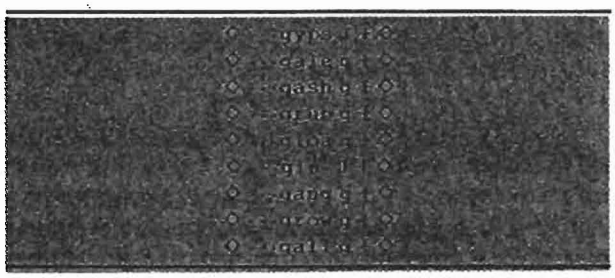
Totaling the weights for 'g' and 'J', we get the following:

The result of this experiment can be summarized as follows: "The correct phoneme is probably 'J', but it might be 'g'."

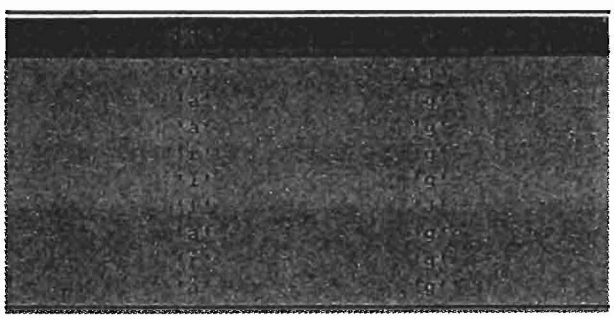
Implementation on the Connection Machine System

Although a full description of implementation of these metrics on the Connection Machine system is beyond the scope of this article, a brief note is in order. The dominant computation in calculating this metric is counting the number of occurrences of each predictor-value/goal-value pair $|D[f = \rho.f][g = \rho.g]|$. This is done in parallel for every possible value of $\rho.f$ and $\rho.g$ using a histogramming algorithm, which takes approximately 16 milliseconds.[13]

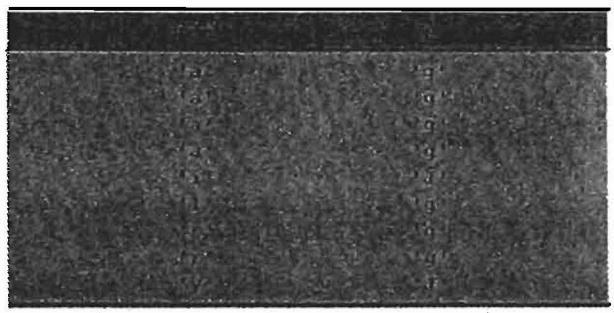
Suppose we have the following database:



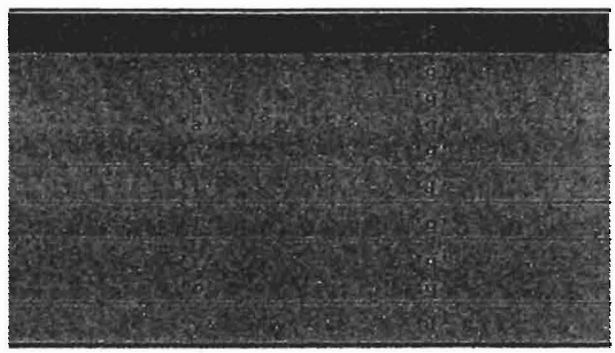
We first select the values of the predictor field $n + 1$ and the goal field p:



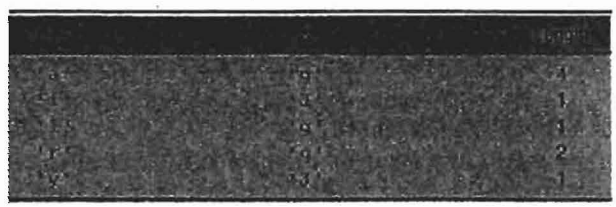
We then sort the records. This takes $\log^2 n$ time:



The next step is to segment the sorted database into regions where both the predictor $(n + 1)$ and the goal (p) are uniform:



Finally, the length of each of these segments is calculated. This takes $\log n$ time:



The time required to perform this set of operations is dominated by the sorting step, which takes $\log^2 n$ time.

The aim of memory-based reasoning is to fill in the goal fields of a target record by retrieving records from a database. The basis of the method is finding the dissimilarity between the target record and each of the data records. The dissimilarity measure is calculated by assigning a weight to each field and a value difference measure to each value occupying those fields. In order to allow for the interaction of pairs of fields, we may employ one of two restriction methods. In predictor restriction, we operate on a subset of the database having the same value as the target for the most important predictor field. In goal restriction, we apply the dissimilarity measure to the

[13] This assumes 16K records on a 16K-processor Connection Machine.

full database to find some plausible values for the goal field, restrict the database to those records having one of these plausible values, and finally apply the dissimilarity measure to this subset. The result of all this is a set of 10 (or so) data records that are similar to the target record, plus dissimilarity measures. The values of their goal fields, weighted according to their dissimilarity, are the plausible values for the goal.

EXPERIMENTAL RESULTS

We started the MBRtalk experiment with a dictionary of slightly over 20,199 words. We randomly selected 4,438 words, with a total of 32,768 letters, for use as the MBRtalk database. The memory-based reasoning algorithm was then applied to 100 words, totaling 772 letters; 86 percent of the phonemes (43 percent of the words) matched the dictionary pronunication exactly. Humans, listening to the output of the program played through a DECtalk® speech synthesizer, judged the pronunciation as being at least as good as the dictionary pronunciation 47 percent of the time, slightly mispronounced 21 percent of the time, and badly mispronounced 32 percent of the time.

For reasons stated previously, the pronunciation task is fundamentally impossible: English pronunciation is too irregular for any sort of inductive reasoning to work all the time. Nevertheless, this is the sort of difficult, noisy real-world problem that we want to be able to solve.

The Representation

When we began work on the pronunciation task, we used a seven-letter window (as in NETtalk). Each window had a central letter, the three preceding letters, the three following letters, the phoneme associated with the central letter, and the stress of the central letter. The seven letter fields were used as predictors, while the phoneme and the stress were used as goals.

After a certain amount of work, we modified this representation. First, we noted that a seven-letter window was inadequate to determine the pronunciation of a word, because to pronounce the first syllable one often needs to know the suffix of the word (e.g., *observatory* = `xbz-Rvxtori` versus *observation* = `cbsxrveSx-n`). Therefore, we extended the window to 15 letters: a central letter, the 4 preceding letters, and the 10 succeeding letters.

In addition, we noted that there were cases where the system produced incoherent values for successive phonemes. For example, the sequence of letters `'qui'` can, depending on whether it is stressed or not, be pronounced either `'Q-I'` or `'k*-'`. In cases where the stress assignment was unclear, it was possible to get incoherent choices, such as `'Q--'` or `'k*I'`. This was resolved by adding new predictor fields: $p-4, \ldots, p-1$, which recorded the preceding phonemes. We then work left to right, filling in appropriate values of these fields as we go. We might start with the following record:

◇ ··· e q uipment··· ···I λ◇

Applying the memory-based reasoning algorithm, we might get either `'k'` or `'Q'` as the correct pronunciation for `'q'`. Assume we get `'Q'`. When we get to the next position in the word, this decision is to be carried along:

◇ ··eq u ipment··· ··IQ λ◇

We now apply the memory-based reasoning algorithm to the letter `'u'` and are assured of getting the only coherent choice, which is `'-'` (silent). The previous decision to pronunce the `'q'` as `'Q'` has effectively determined the pronunciation of `'u'`.

This change of representation improved the coherence of MBRtalk pronunciations, the lesson being that in many cases all we need do to improve system performance is to improve the representation; reprogramming or modification of an existing rule base may not be necessary.

The Experiment

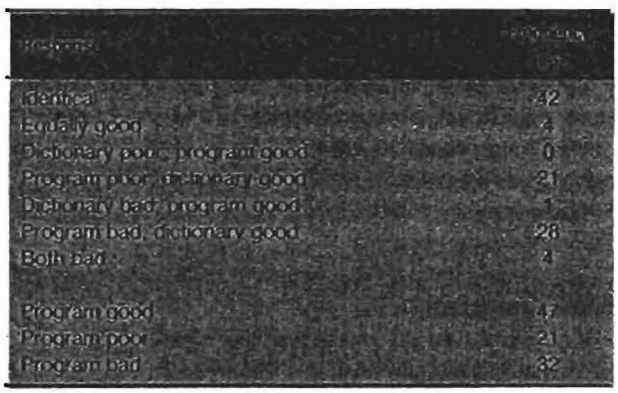
We now come to the primary experiment. As explained above, we used a 4438-word subset of *Merriam-Webster's Pocket Dictionary* plus the memory-based reasoning algorithm to pronounce 100 novel words. We then compared MBRtalk pronunciations against the dictionary's. The strictest possible standard of comparison is to require the two to be identical. Forty-three percent of the words met this standard (86 percent of the phonemes).

A more realistic standard of comparison, however, is the judgment of a native speaker of English. (The output of the program, as evaluated by human judges, is shown in Appendix B, p. 1226.) Many words have several acceptable pronunciations. For example, the word *object* may be pronounced with the stress on either the first or the second syllable. In such cases, the listeners were requested to mark the two pronunciations "equally good." Also, in some contexts two phonemes are indistinguishable. For example, it is not possible to hear the difference between the dictionary pronunciation of *furniture* (`'f-RnIC-R-'`) and that produced by memory-

DECtalk is a trademark of Digital Equipment Corporation.

based reasoning (`'f-RnxC-R-'`). In these cases, the listeners marked the two pronunciations "identical."

For this experiment, the program's pronunciation and the dictionary's pronunciation were used to drive a DECtalk speech synthesizer. The order of presentation was randomized, so that the judges could not tell which was which. They had several possible replies: They could state that there was no audible difference between the two pronunciations, that the two pronunciations were different but equally good, that one was slightly mispronounced, or that either or both words were badly mispronounced. They could listen to the two words as many times as was necessary to make a decision. The following table summarizes the results of the experiment. First, we have the percentage of the words assigned by a plurality of the judges to each category. Then we have the responses tabulated according to whether the program's pronunciation was judged good, poor, or bad.



Analysis

Some of the errors can be attributed to having insufficient information to choose the correct pronunciation. Pronunciation can depend on whether a word is a noun or a verb, which is information that the program does not have available. For example, the program takes the word *abject* and pronounces it as if it were a verb (`'xbjEkt'`). Of the 53 mispronunciations, 19 were due to this cause. In addition, foreign words that have been incorporated into English are often pronounced according to the rules of the language from which they came. Thus, the program pronounces the word *montage* as if it were English (`'mantIJ-'`). There were 6 errors in this category.

There were three errors due to the program encountering a rare combination of letters, such as a word-initial `'ps'` (psychiatry).

In summary, out of the 100 words given to the memory-based reasoning algorithm, 47 were pronounced correctly, 21 were slightly mispronounced, and 32 were badly mispronounced. Of the 53 words that were mispronounced, 19 were due to the lack of part-of-speech information, 6 were because the word was from a foreign language, and 3 were because they contained rare combinations of letters. Thus, the reasoning procedure made 28 errors that it might possibly have avoided.

Considering the difficulty of the task, we believe this level of performance to be highly encouraging. Further work in the pronunciation domain will concentrate on improving the representation (e.g., adding part-of-speech information) and on isolating the reasons for avoidable errors.

CONCLUSIONS AND PROSPECTS

Memory-based reasoning fits the Connection Machine system well because both the cost of the hardware and the speed of execution for the algorithms scale well with increasing database size: The cost of the hardware is $O(n \log n)$ in the size of the database, while the processing time is $O(\log^2 n)$.

On a sequential machine, both the cost of the hardware and the time required for processing grow linearly with the size of the database. The linear growth in processing time causes the time to make a decision to become unacceptably large for sizable databases. One solution to this is to build faster sequential machines. However, there are absolute limits to how fast a sequential machine can be, and as those limits are approached, the hardware becomes extremely expensive.

Another solution is to use a vector machine. Vector machines may work well for some aspects of memory-based reasoning. However, unless we allow the number of processors utilized to grow as the size of the database increases, we get no more than a constant speedup over a sequential machine.

If we allow the number of processors to grow as the size of the database increases, then we have a computer that is in accordance with our expectations for a truly parallel machine. At this point, interprocessor communication becomes an issue. A highly interconnected communication scheme, such as a hypercube or butterfly, is necessary to allow the counting operations to be performed in a reasonable amount of time ($\log^2 n$).

One issue remains: whether to use a fine-grained SIMD machine or a large-grained MIMD machine. The memory-based reasoning algorithms are intrinsically SIMD, so there is no need for the added circuitry necessary to support MIMD. In addition, the memory-based reasoning algorithms are communi-

cation intensive. A fine-grained architecture will have more communication nodes and a greater total communication bandwidth than a large-grained architecture with the same amount of hardware.

Thus, the ideal machine for memory-based reasoning would have a parallel, highly interconnected, fine-grained SIMD architecture.

In the long run, we believe that memory-based reasoning systems can play an important role in building truly intelligent systems, but we have much to do and learn before such a goal can be achieved. In the short run, memory-based reasoning systems should be able to function like expert systems in domains where there are databases of situations and outcomes/actions. Possible application areas include medical diagnosis and proposal of diagnostic tests, weather forecasting, credit decision support, investment advising, and insurance risk assessment.

There are many advantages:

- No expert is required; a system builder need only identify and mark database contents according to whether they are symptoms or features of situations, outcomes or actions, or optional tests that can be ordered.
- Memory-based systems can form hypotheses on the basis of even a single precedent, something rule-based systems cannot do—rules are inherently summaries of regularities among classes of items.
- Memory-based reasoning systems "know when they don't know": if no items in memory are closely related to the current item being analyzed, then the system will be able to recognize this fact and tell the user. This fact should also allow memory-based reasoning systems to degrade gracefully.
- Memory-based systems should require far less time to implement. However, once constructed, rule-based systems can operate on much less powerful hardware. Thus we also intend to explore the automatic generation of rules by memory-based systems.

Computer learning has generally stayed within the heuristic search/deductive reasoning paradigm: Most learning algorithms try to derive a set of rules from a body of data. Memory-based reasoning, in its pure form, differs from traditional learning methods in that no rules are ever created. This has three advantages:

- Deductions made by the system are achieved without the intermediate use of rules as a representation, so there are fewer opportunities for inadvertently introducing inaccuracies, such as those that result from the combining "confidence levels" for a chain of rules.
- Rule generation has a combinatorially explosive search space of possible rules to contend with, while memory-based systems can focus their analysis on the cases at hand. It may be that, to capture a large knowledge base, so many rules are needed that the rule-generation process will never finish; there is never any certainty that the resulting rule set is complete, and the resulting system may fail at any moment. With a memory-based system, the data used as the basis of reasoning are always available. Therefore, within limits imposed by the accuracy of the retrieval mechanism and the contents of the database, errors of omission should not occur.
- Rule-based systems often include invented variables to insure proper sequencing of rules. Ordering problems and unexpected interactions are not a difficulty in memory-based reasoning systems.

Both learning/rule-based systems and memory-based systems are fundamentally limited by the databases they work from and by their retrieval methods. There will be cases where a database is too sparse to give meaningful answers, and there may be cases where spurious correlations will give bad answers. In both classes of systems, statistical tests can guard against these problems. For both systems, there may also be cases where the recall mechanism fails to find all the relevant data. It is necessary to take steps to ensure such recall failures do not introduce systematic biases.

It isn't quite fair to compare connectionist research with memory-based reasoning, since the goals of each are so different. There are, however, several inherent advantages to memory-based reasoning:

- Memory-based systems require no training; they begin to produce results immediately.
- Memory-based systems are understandable and can potentially explain why they act as they do; foreseeable connectionist models will all be very difficult to analyze.
- Since connectionist models average the values of features, they may permanently lose some information that is present in the particular sets of features that co-occur in episodes. Memory-based systems may be inherently more powerful.

We intend to pursue this research on real medical, weather, and financial databases, and look forward with excitement to the future of this technology.

APPENDIX A. PRONUNCIATION KEY

The following is a key to the phonetic alphabet used in this article. Pronunciations are as in *Merriam-Webster's Pocket Dictionary*.

#	auXiliary	f	Fight	N	adjacENt	U	ambUsh
!	marZipan	G	aloNG	n	Neat	u	tOO
*	anyOne	g	Great	O	annOY	v	aboVe
@	abAck	h	Hot	o	cOld	W	OUt
A	fIle	I	bIt	p	Pet	w	Wet
a	cAr	i	bEAt	Q	QUaint	X	affiX
b	Bat	J	Jet	R	accuRacy	x	Allow
C	CHurch	K	anXious	r	Red	Y	cUte
c	OUght	k	Cat	S	daSH	y	Yet
D	baTHe	L	abLE	s	Sleep	Z	aZure
d	Dull	l	Lead	T	THink	z	siZe
E	bEd	M	albinisM	t	Tin	↑	resUlt
e	mAke	m	Meat				

APPENDIX B. MBRtalk Experimental Results

The following 41 words were performed correctly by MBRtalk:

achromatic	@k-rxm@tIk	@k-rxm@tIk	latchet	l@-C-xt	l@-C-xt
admonition	@dmxnISx-n	@dmxnISx-n	leaf	li-f	li-f
angry	@Ggri	@Ggri	masquerade	m@skx-red-	m@skx-red-
arable	@rxbL-	@rxbL-	mate	mate	met-
ballad	b@l-xd	b@l-xd	mull	m-L-	m-L-
carnivorous	karnIvxrx-s	k@rnIvxrx-s	munificent	mYnIfxsxnt	mxnIfxsxnt
celery	sElxri	sElxri	natatorium	netxtorixm	netxtorixm
clung	klxG-	klxG-	numerable	nYmxrxbL-	nYmxrxbL-
coalfield	ko-lfi-ld	ko-lfi-ld	parquetry	parkx-tri	parkx-tri
confidential	kanfxdEnC-L	kanfIdEnC-L	party	parti	parti
contrite	kantrAt-	kantrAt-	pepsin	pEpsxn	pEpsxn
drum	drxm	drxm	pretty	prIt-i	prEt-i
enigmatic	EnIgm@tIK	EnIgm@tIk	pride	prAd-	prAd-
eternal	It-Rn-L	It-Rn-L	qualification	Q-alxfxkeSx-n	Q-alxfxkeSx-n
forlorn	f-Rlcrn	forlcrn	round	rW-nd	rW-nd
formative	fcrmxtIv-	fcrmxtIv-	scaleless	skel-lxs-	skel-lxs-
frappe	fr@p-	fr@p-	soothsayer	su-T-se-R	su-T-se-R
furniture	f-RnIC-R-	f-RnxC-R-	stupefy	stYpxfA	stYpxfA
furrier	f-R-i-R	f-R-i-R	symbolization	sImbxlxzeSx-n	sImbxlxzeSx-n
gratification	gr@txfxkeSx-n	gr@txfxkeSx-n	tester	tEst-R	tEst-R
grin	grIn	grIn	veronica	vxranIkx	vxranIkx
impossibly	Impas-xbli	Impas-xbli	whitish	w-AtIS-	w-AtIS-
inebriate	Inibrixt-	InEbriet-	woodpecker	wU-dpEk-R	wU-dpEk-R
kiss	kIs-	kIs-			

The following two words were judged "somewhat mispronounced" by MBRtalk and are from foreign languages:

micron	mAkran	mIkrxn
viola	violx	vAxlx

The following 12 words were judged "somewhat mispronounced" with the error attributable to stress misassignment. Assigning stress to the wrong syllable often results in the wrong phoneme being chosen.

TABLE II.

arteriosclerosis	artIriosklxrosxs	xrtErixsklErxsxs
barroom	barr-um	b@r-x-m
broider	brO-d-R	bru-d-R
gadfly	g@dflA	g@d-li
instrumental	InstrxmEnt-L	InstrYmxnt-L
item	Atxm	It-m
jeremiad	JErxmAxd	Jxrimi-d
libel	lAb-L	lIb-L
outmoded	W-tmodxd	W-tmod-d
outplay	W-tple-	W-tpl-i
photosensitive	f-otxsEnsxtIv-	f-atxsinsxtIv-
preoccupation	priak-YpeSx-n	pri-k-upeSx-n
refill	rifil-	rEfil-
rhinoceros	r-Anasxrxs	r-Inxsxrcs
surveillance	s-Rve-l-Yns-	s-Rv-llxns-
viscera	vIs-xrx	vIs-R-
who	-hu	w--

undercut	xnd-Rkxt	xnd-Rkxt
abject	@bJEkt	xbJEkt
action	@kSx-n	@kSx-n
archivist	ark-xvxst	ark-Ivxst
compress	kamprEs-	kamprxs-
concentration	kansxntreSx-n	knxsEntreSx-n
definition	dEfxnISx-n	dIfInxSx-n
redistrict	ridIstrIkt	ridistrIkt
molest	mxlEst	molxst
outguess	W-tg-Es-	W-tgx-s-
perfumery	p-RfYmxri	p-RfYmxri
projectile	prxJEkt-L-	proJxkt-L-

The following seven words were judged "somewhat mispronounced," with no obvious explanation for the failure.

amphibian	@mf-Ibixn	xmf-Ibx-n
bosom	bUzxm	bosxm
familial	fxmIly-L	fxmIlx-L
labor	leb-R	l@b-R
middy	mId-i	mAd-i
vinyl	vAn-L	vAniL
vitalize	vAt-LAz-	vIt-LAz-

The following four words were judged "badly mispronounced" and come from foreign languages.

menage	menaZ-	mEnIJ-
montage	mantaZ-	mantIJ-
ricochet	rIkxS-e-	rIkxS-xt
sardine	sardin-	sardxn-

The following three words were judged "badly mispronounced" and contain rare letter/sound combinations.

aweigh	-ue---	c-----
suture	suC-R-	sxt-R-
psychiatry	-sxk-Axtri	psIC-ixtri

The following seven words were judged "badly mispronounced," with the error attributable to stress misassignment. Assigning stress to the wrong syllable often results in the wrong phoneme being chosen.

aloft	xlcft	@lxft
bedraggled	bIdr@g-L-d	bEd-xg-L-d
consent	kxnsEnt	kansxnt
humorous	hYmxrx-s	hxmorx-s
marina	mxrinx	m@rxnx
mutilator	mYt-Let-R	mYtIlxtor
relax	rIl@X	rElxX

The 18 words in Table II were judged "badly mispronounced," with no obvious explanation for the failure.

For the following word, MBRtalk's pronunciation and the dictionary's pronunciation are identical, and both were judged "badly mispronounced." Either the DECtalk or our phonetic alphabet is inadequate to capture the "correct" pronunciation.

dumpy	dxmpi	dxmpi

Acknowledgments. Many thanks to George Robertson and Guy Steele for reading drafts of this article; to the Connection Machine System Group for providing the hardware; to Charlie Rosenberg for providing us with the database; to Jim Bailey, Gregory Wilcox, Brewster Kahle, Sarah Ferguson, Carl Feynman, and Danny Hillis for listening to MBRtalk output and evaluating it; to Chris Maruk for helping prepare the manuscript; and to Robert Thau and Donna Fritzsche for general assistance.

REFERENCES

1. Alterman, R. Issues in adaptive planning. Rep. UCB/CSD 87/304, Computer Science Division, Univ. of California at Berkeley, July 1986.
2. Charniak, E. The Bayesian basis of common sense medical diagnosis. In *Proceedings of the 3rd National Conference on Artificial Intelligence (AAAI-83)* (Washington, D.C., Aug. 22–26). American Association for Artificial Intelligence, Menlo Park, Calif., 1983, pp. 70–73.
3. Cheeseman, P. A method for computing Bayesian probability values for expert systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, West Germany, Aug. 8–12). 1983, pp. 198–202.
4. DeJong, G., and Mooney, R. Explanation-based learning: An alternative view. *Mach. Learn. 1*, 2 (Apr. 1986), 145–176.
5. Dreyfus, H., and Dreyfus, S. Why computers may never think like people. *Technol. Rev. 89*, 1 (Jan. 1986), 42–61.
6. Fahlman, S.E. *NETL: A System for Representing and Using Real-World Knowledge.* MIT Press, Cambridge, Mass., 1979.
7. Feldman, J.A. Introduction to the special issue on connectionism. *Cognitive Sci. 9*, 1 (Jan.–Mar. 1985), 1–169.
8. Feldman, J.A., and Ballard, D.H. Connectionist models and their properties. *Cognitive Sci. 6*, 3 (July–Sept. 1982), 205–254.
9. Gelerntner, H. Realization of a geometry theorem proving machine. In *Computers and Thought*, E.A. Feigenbaum and J. Feldman, Eds. McGraw-Hill, New York, 1963.
10. Hewitt, C. Description and theoretical analysis of PLANNER. Doctoral dissertation, Dept. of Mathematics, MIT, Cambridge, Mass., 1972.
11. Hillis, D. *The Connection Machine.* MIT Press, Cambridge, Mass., 1985.
12. Hillis, W.D., and Steel, G.L., Jr. Data parallel algorithms. *Commun. ACM 29*, 12 (Dec. 1986).
13. Hinton, G., and Anderson, J., Eds. *Parallel Models of Associative Memory.* Lawrence Erlbaum Associates, Hillsdale, N.J., 1981.
14. Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Mich., 1975.
15. Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. 79* (1982), 2554–2558.
16. Kanerva, P. Self-propagating search: A unified theory of memory. Rep. CSLI-84-7, Center for the Study of Language and Information, Stanford University, Calif., Mar. 1984.
17. Kolodner, J. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Lawrence Erlbaum Associates, Hillsdale, N.J., 1984.
18. Kowalski, R.A. Predicate logic as programming language. In *Proceedings of the IFIPS Congress* (Amsterdam). International Federation of Information Processing Societies, 1974, pp. 570–574.
19. Lebowitz, M. Integrated learning: Controlling explanation. *Cognitive Sci. 10*, 2 (Apr.–June 1986), 219–240.
20. Lebowitz, M. Not the path to perdition: The utility of similarity-based learning. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)* (Philadelphia, Pa., Aug. 11–15). American Association for Artificial Intelligence, Menlo Park, Calif., 1986, pp. 533–537.
21. Lee, W. *Decision Theory and Human Behavior.* Wiley, New York, 1971.
22. McClelland, J.L., and Rumelhart, D.E., Eds. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, Mass., 1986.
23. *Merriam-Webster's Pocket Dictionary.* Merriam-Webster, Springfield, Mass., 1974.
24. Michalski, R., Carbonell, J., and Mitchell, T., Eds. *Machine Learning.* Tioga, Palo Alto, Calif., 1983.
25. Michalski, R., Carbonell, J., and Mitchell, T., Eds. *Machine Learning.* Vol. 2, Tioga, Palo Alto, Calif., 1986.
26. Miller, G.A., Gelenter, E., and Pribram, K. *Plans and the Structure of Behavior.* Holt, Rinehart and Winston, New York, 1960.
27. Minsky, M. *The Society of Minds.* Simon and Schuster, New York: To be published.
28. Mitchell, T., Utgoff, P., and Banerji, R. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine Learning*, R. Michalski et. al., Eds. Tioga, Palo Alto, Calif., 1983.
29. Mooney, R. A domain independent explanation-based generalizer. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)* (Philadelphia, Pa., Aug. 11–15). American Association for Artificial Intelligence, Menlo Park, Calif., 1986, pp. 551–555.
30. Newell, A. The knowledge level. *AI Mag. 2*, 2 (Summer 1981), 1–20.
31. Newell, A., and Simon, H. *Human Problem Solving.* Prentice-Hall, Engelwood Cliffs, N.J., 1972.
32. Newell, A., and Simon, H.A. GPS, a program that simulates human thought. In *Computers and Thought*, E.A. Feigenbaum and J. Feldman, Eds. McGraw-Hill, New York, 1963, pp. 279–293.
33. Newell, A., Ernst, G., and Shaw, J. Elements of a theory of human problem solving. *Psychol. Rev. 65*, (1958), 151–166.
34. Quinlan, J.R. Discovering rules from large collections of examples: A case study. In *Expert Systems in the Micro Electronic Age*, D. Michie, Ed. Edinburgh University Press, Edinburgh, 1979.
35. Robinson, J.A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23–41.
36. Rosenblatt, F. *Principles of Neurodynamics: Perceptions and Theory of Brain Mechanisms.* Spartan, Washington, D.C., 1961.
37. Rumelhart, D.E., Hinton, G.E., and Williams, R.J. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, J.L. McClelland and D.E. Rumelhart, Eds. MIT Press, Cambridge, Mass., 1986.
38. Samuel, A. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev. 3*, 3 (1959). 210–299.
39. Samuel, A. Some studies in machine learning using the game of checkers II. *IBM J. Res. Dev. 11*, 6 (1967), 601–617.
40. Schank, R.C., *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, New York, 1982.
41. Sejnowski, T.J., and Rosenberg, C.R. NETtalk: A parallel network that learns to read aloud. Electrical Engineering and Computer Science Dept., Johns Hopkins Univ. Tech. Rep. JHU/EECS-86/01, 1986.
42. Shortliffe, E. *Computer Based Medical Consultations: MYCIN.* Elsevier North-Holland, New York, 1976.
43. Stanfill, C., and Kahle, B. Parallel free-text search on the Connection Machine system. *Commun. ACM 29*, 12 (Dec. 1986), 1229–1239.
44. Sussman, G., Winograd, T., and Charniak, E. MICRO-PLANNER reference manual. AI Memo 203a, MIT AI Laboratory, MIT, Cambridge, Mass., Dec. 1971.
45. Waltz, D.L., Genesereth, M., Hart, P., Hendrix, G., Joshi, A., McDermott, J., Mitchell, T., Nilsson, N., Wilensky, R., and Woods, W. Artificial intelligence: An assessment of the state-of-the-art and recommendation for future directions. *AI Mag. 4*, 3 (Fall 1983), 55–67.
46. Winston, P. Learning structural descriptions from examples. In *The Psychology of Computer Vision*, P. Winston, Ed. McGraw-Hill, New York, 1975.
47. Woods, W.A. Transition network grammars for natural-language analysis. *Commun. ACM 13*, 10 (Oct. 1970), 591–606.

CR Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*single-instruction-stream, multiple-data-stream processors (SIMD)*; I.2.0 [**Artificial Intelligence**]: General—*paradigms*; I.2.6 [**Artificial Intelligence**]: Learning—*induction*
General Terms: Algorithms, Theory
Additional Key Words and Phrases: Connection Machine, massively parallel architectures, memory

Authors' Present Address: Craig Stanfill and David Waltz, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142-1214. David Waltz is also at Computer Science Dept., Brandeis Univ., Waltham, MA 02254.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.