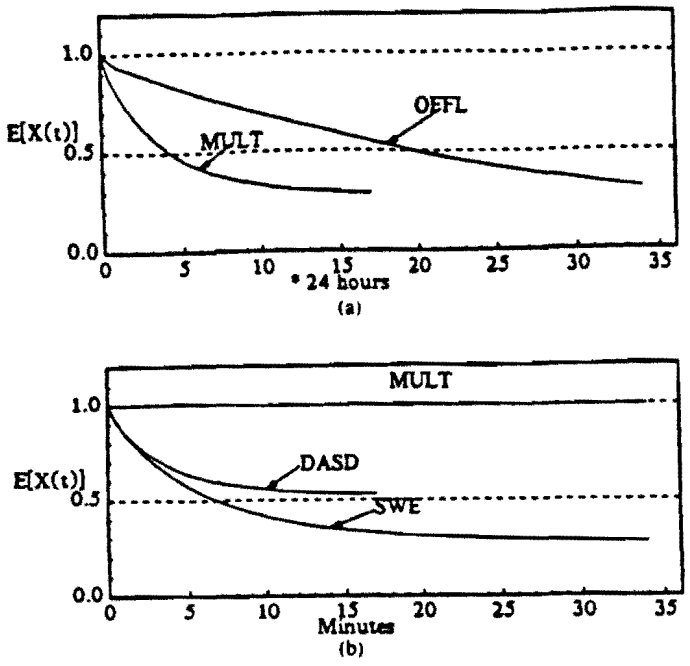


Fig. 7. Expected reward rate, $E[X(t)]$.

components (hardware and software) is reflected in a multiple error model. The proposed reward measure allows us to predict the performability of the system based on the service and error rates. It is suggested that other production systems be similarly analyzed so that a body of realistic data on computer error and recovery models is available.

ACKNOWLEDGMENT

The authors would like to thank J. Gerardi and the members of Reliability, Availability and Serviceability Group at IBM-Poughkeepsie for access to the data and for their valuable comments and suggestions. Thanks are also due to Dr. W. C. Carter for valuable discussions during the initial phase of this work. We also thank L. T. Young and P. Duba for their careful proofreading of the draft of this manuscript. Finally, we thank the referees for their constructive comments which were extremely valuable in revising the paper.

REFERENCES

[1] J. F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.*, pp. 648-657, July 1982.
[2] R. M. Geist, M. Smotherman, K. S. Trivedi, and J. Bechta Dugan, "The reliability of life critical systems," *Acta Informatica*, vol. 23, pp. 621-642, 1986.
[3] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi, "The system availability estimator," in *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, Vienna, Austria, July 1986.
[4] O. Schoen, "On a class of integrated performance/reliability models based on queuing networks," in *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, Vienna, Austria, July 1-4, 1986, pp. 90-95.
[5] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, pp. 214-237, Aug. 1986.
[6] B. Littlewood, "Theories of software reliability: How good are they and how can they be improved?" *IEEE Trans. Software Eng.*, vol. SE-6, pp. 489-500, Sept. 1980.
[7] M. C. Hsueh and R. K. Iyer, "A measurement-based model of software reliability in a production environment," in *Proc. 11th Annu. Int. Comput. Software Appl. Conf.*, Tokyo, Japan, Oct. 7-9, 1987, pp. 354-360.
[8] X. Castillo, "A compatible hardware/software reliability prediction model," Ph.D. dissertation, Carnegie-Mellon Univ., July 1981.
[9] D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[10] H. Spath, *Cluster Analysis Algorithms*. West Sussex, England: Ellis Horwood, 1980.
[11] M. C. Hsueh, "Measurement-based reliability/performability models," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois at Urbana-Champaign, Aug. 1987.
[12] IBM Corp. *Environmental Record Editing & Printing Program*, IBM Corp., 1984.
[13] R. K. Iyer, L. T. Young, and V. Sridhar, "Recognition of error symptoms in large systems," in *Proc. 1986 IEEE-ACM Fall Joint Comput. Conf.*, Dallas, TX, Nov. 2-6, 1986, pp. 797-806.
[14] R. A. Howard, *Dynamic Probabilistic Systems*. New York: Wiley, 1971.
[15] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[16] R. M. Smith and K. S. Trivedi, "A performability analysis of two multiprocessor systems," in *Proc. 17th Int. Symp. Fault-Tolerant Comput.*, Pittsburgh, PA, July 6-8, 1987, pp. 224-229.
[17] R. A. Sahner and K. S. Trivedi, "Reliability modeling using SHARPE," *IEEE Trans. Reliability*, pp. 186-193, June 1987.

88-19

Distributed and Fault-Tolerant Computation for Retrieval Tasks Using Distributed Associative Memories

JOIS MALATHI CHAR, VLADIMIR CHERKASSKY, HARRY WECHSLER, AND GEORGE LEE ZIMMERMAN

***Abstract*—We suggest the distributed associative memory (DAM) model for distributed and fault-tolerant computation as related to retrieval tasks. The fault tolerance is with respect to noise in the input key data and/or local and global failures in the memory itself. We have developed working models for fault-tolerant image recognition and database information retrieval backed up by experimental results which show the feasibility of such an approach.**

***Index Terms*—Database retrieval, distributed associative memory (DAM), distributed computation, fault tolerance, neural networks, recognition.**

I. INTRODUCTION

Artificial Intelligence (AI) deals with the types of problem solving and decision making that humans continuously face in dealing with the world. Such activity involves by its very nature complexity, uncertainty, and ambiguity, all of which can distort the phenomena being analyzed. However, following the human example, any corresponding computer system should process information such that the results are invariant to the vagaries of the data acquisition process. Furthermore, one would expect such computer systems to be fault tolerant, i.e., to display robustness, if and when some of their hardware were to fail. We suggest in this paper how a particular type

Manuscript received April 15, 1987; revised October 15, 1987. This work was supported in part by the Graduate School at the University of Minnesota (J. M. Char and V. Cherkassky), by the National Science Foundation under Grant EET-8713563 to H. Wechsler, and by the Microelectronics and Information Science Center (MEIS) of the University of Minnesota (G. L. Zimmerman).

J. M. Char is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455

V. Cherkassky, H. Wechsler, and G. L. Zimmerman are with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455

IEEE Log Number 8719338.

of distributed and fault-tolerant computation modeled after the distributed associative memory (DAM) [14] paradigm can be made useful for tasks such as those encountered in object recognition and database retrieval.

One can consider intelligent behavior as an information processing task and then address the three levels at which such tasks must be understood [17]. First, the basic computational theory specifies what is the task, why is it appropriate, and what is the strategy by which it can be carried out. Second, the representation and algorithm specify how the computational theory can be implemented in terms of input, output, and transformations. Third, the hardware specifies the actual implementation. It is clear that the task determines the mixture of representations and algorithms. It is also clear that a good fit/match among the three levels is highly desirable and beneficial. In our case, we define our task as fault-tolerant data retrieval and suggest to solve it as a constraint optimization problem. The representation and algorithms to be discussed in detail later on are characteristic of an emerging AI trend, that of parallel and distributed computation. Characteristic to such a trend are neural networks (NN's) also known as ANS/Artificial Neural Systems [7].

According to Sejnowski [16], NN's allow objects and their relationships to be internally represented by attractors (i.e., fixed-points), which makes the search for the best match between the world and the internal representation of the world (encoded through the dynamics of the network) much more powerful than previous template matching. Another aspect which makes NN's attractive comes from the ability to implement them as adaptive systems, able to learn and self-organize. The lack of a meaningful learning capability is one of the major drawbacks for AI.

It is worthwhile to briefly survey the origins of NN's. The idea of a distributed network for computational tasks like recognition was suggested by Rosenblatt [19] in the form of the two-layer Perceptron. Minsky and Papert [18] showed that such networks could fail on even simple tasks like implementing the XOR function. Scientists do not give up, even more so when there is a theorem by Kolmogorov stating that any continuous function of n variables can be computed using a three-layer Perceptron with $n(2n + 1)$ nonlinear nodes [15]. Unfortunately, the theorem is not of the constructive type. Most recently, connectionist models were suggested in the form of multilayer networks with hidden units. Regarding the learning algorithm, backpropagation learning was introduced by Rumelhart *et al.* [16]. Even though such an algorithm does not enjoy proof of convergence, it still proved very successful in a system NETtalk developed by Sejnowski [21] for word pronunciation. The connectionist models could be implemented in parallel (SIMD) on a Connection Machine [9] running under the "marker-passing type" mode [4]. Using such an approach, Stanfill and Waltz [22], suggest the idea of memory-based reasoning (similarity-based induction) and were able to duplicate to a close approximation the results obtained by Sejnowski. The DAM paradigm to be introduced in the next section is a known mathematical tool. It enjoys the benefit of a convergent learning procedure like the Widrow-Hoff algorithm [14].

Neural networks were developed originally to account for biological memory systems. They implement a type of distributed representation and computation, where a large number of highly interconnected "simple" processing elements (PE) operate in parallel. NN's can be structured both hierarchically and heterarchically, and through the use of efferent, afferent, lateral connections, and/or hidden layers, the limitations due to local processing and lack of feedback could be removed [7], [10]. NN's provide a good fit among the three levels at which recognition tasks should be understood. Through the NN's collective dynamics, the emergent behavior which is the result of both competition and cooperation between neighboring PE's yields the optimal solution subject to contextual constraints implicitly embedded in the net of interconnections. Such an approach is akin to relaxation [17].

The generic recognition process matches a derived (invariant) input representation of the short-term memory (STM) type against LTM (long-term memory). A synergetic approach suggests the LTM organization in terms of DAM's. Specifically, the DAM's are related to GMF (generalized match filters) [2] and SDF (synthetic discriminant functions) [8]. Like them, the DAM's attempt to capture a distributed representation which averages over the variations belonging to the same class. A parallel and distributed mode of computation for visual tasks is further suggested by the retinotopic and parallel cytoarchitecture characteristic to the visual cortex [11]. Such distributed representations are also consistent with Gestalt (holistic) recognition, where the holistic organization can be imposed over space and/or time. The DAM's allow for the implicit representation of structural relationships and contextual information. Finally, because information is distributed in the memory, the overall function of the memory becomes resistant to noise, faults in memory, and degraded stimulus key vectors. A particular analogy might be useful. The difference between addressed computer memories and DAM's is similar to the difference between a photograph and a hologram. If a photograph is cut in half and one half is thrown away, fifty percent of the information is lost, and the information lost is that from the half thrown away. In a similar manner, if half of the address for a normal computer memory is thrown away, it would be very had to find that memory location. On the other hand, if a hologram is cut in half, again fifty percent of the information held by the whole hologram will be lost, but the whole image can be reconstructed, although the reconstructed image will be noisier. This, in essence, is the way the DAM behaves.

II. DAM Background

We describe first the DAM modeling in terms of representation and algorithms and then we discuss the mathematics involved in computing the generalized inverse as required by such modeling. There are two main phases in the operation of DAM's:

1) *Memory Construction Phase:* when the memory matrix is created from a given set of associations.

2) *Recall Phase:* (i.e., memory indexing and retrieval), when a noisy (or perfect) version of a stimulus vector that is part of a stored association is used to retrieve its associated vector (or a close approximation of it).

Specifically the DAM allows for memory to be reconstructive, i.e., it yields the entire output vector (or a closed approximation of it) even if the input is noisy, partially present, or if there is noise in the memory itself (computer components like neurons are not always reliable and may fail too!).

The memory matrix for the autoassociative case is evaluated using the equation given below

$$M = SS^{+}$$

where M is the memory matrix and S^{+} is the generalized inverse of the input matrix S (each column of S corresponds to one stimulus vector). The generalized inverse of a nonsquare matrix is a generalization of the ordinary inverse for square nonsingular matrices [20]. A heteroassociative model can be easily developed assuming a memory of type $M = RS^{+}$ where $S \neq R$. S and R are called the forcing (stimulus matrix) and coupling (response matrix) functions, respectively.

The retrieval operation for an autoassociative case, i.e, the attempt to recall memory contents for an input key t is shown below.

$$\hat{s} = Mt$$

where $\hat{s}$ is the recalled approximation of the association. The recall operation for the heteroassociative case can retrieve either the forcing (stimulus) function s or the coupling (response) r. Specifically if the corresponding matrices are S and R, respectively, then

a) if $t \in \{r\}$ then $M = SR^{+}$, $\hat{s} = Mt$.
b) if $t \in \{s\}$ then $M = RS^{+}$, $\hat{r} = Mt$.

The memory matrix M defines the space spanned by the input

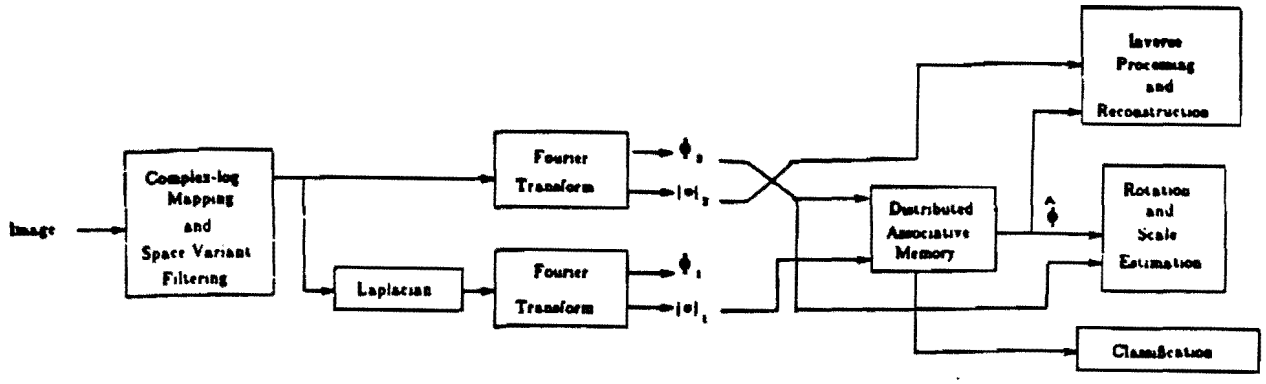


Fig. 1. Block diagram of the system.

representations. The recall operation is implemented through the use of M as a projection operator. The retrieved data $\hat{s}$ (for case b) obeys the relationship

$$t = \hat{s} + \tilde{s}$$

where $\hat{s}$ plays the role of an optimal associative recollection/linear regression in terms of M. The residual $\tilde{s}$ is orthogonal to the space spanned by M and represents the novelty in t with respect to the stored data in M. It is the decomposition of t in terms of $(\hat{s}, \tilde{s})$ that can be used to facilitate the learning of new things.

The computationally intensive module needed to compute the generalized inverse of a matrix [20] implements a least square solution of the Gram-Schmidt orthogonalization process. Our simulation experiments implement such an approach and are described in the next two sections.

III. Invariant Image Recognition

The challenge of the visual recognition problem stems from the fact that the projection of an object onto an image can be confounded by several dimensions of variability such as uncertain perspective, changing orientation and scale, sensor noise, occlusion, and nonuniform illumination. A vision system must not only be able to sense the identity of an object despite this variability, but must also be able to characterize such variability—because the variability inherently carries much of the valuable information about the world. Our goal is to derive the functional characteristics of image representations suitable for invariant recognition using a distributed associative memory. The main question is that of finding appropriate transformations such that interactions between the internal structure of the resulting representations and the distributed associative memory yield invariant recognition. This should be contrasted to earlier attempts to use DAM's directly on raw input data [14].

We approach the problem of object recognition with three requirements: classification, reconstruction, and characterization. Classification implies the ability to distinguish objects that were previously encountered. Reconstruction is the process by which memorized images can be drawn from memory given a distorted version exists at the input. Characterization involves extracting information about how the object has changed from the way in which it was memorized. Our goal is to discuss a system which is able to recognize memorized two-dimensional objects regardless of geometric distortions like changes in scale and orientation, and can characterize those transformations. The system [25] also allows for noise and occlusion and is tolerant of memory faults.

We shortly examine the various components used to produce the vectors which are associated in the distributed associative memory. The block diagram which describes the various functional units involved in obtaining an invariant image representation is shown in Fig. 1. The image is complex-log conformally mapped. This transformation maps the Cartesian space into the log-polar domain such that rotation and scale changes become translation in the transform domain. Along with the conformal mapping, the image is also filtered by a space variant filter to reduce the effects of aliasing. The conformally mapped image is then processed through a Laplacian in order to solve some problems associated with the conformal mapping. The Fourier transform of both the conformally mapped image and the Laplacian processed image produce the four output vectors. The magnitude output vector $|\bullet|_1$ is invariant to linear transformations (scale and rotation) of the object in the input image.

We now discuss the result of computer simulations of our system. Images of objects are first preprocessed through the system outlined in Fig. 1. The output of such a subsystem is four vectors $|\bullet|_1$, Φ_1, $|\bullet|_2$, and Φ_2. We construct the memory by associating the stimulus vector $|\bullet|_1$ with the response vector Φ_2 for each object in the database. To perform a recall from the memory, the unknown image is preprocessed by the same subsystem to produce the vectors $|\tilde{\bullet}|_1$, $\tilde{\Phi}_1$, $|\tilde{\bullet}|_2$, and $\tilde{\Phi}_2$. The resulting stimulus vector $|\tilde{\bullet}|_1$ is projected onto the memory matrix to produce a response vector which is an estimate of the memorized phase $\hat{\Phi}_2$. The estimated phase vector $\hat{\Phi}_2$ and the magnitude $|\tilde{\bullet}|_1$ are used to reconstruct the memorized object. The difference between the estimated phase $\hat{\Phi}_2$ and the unknown phase $\tilde{\Phi}_2$ is used to estimate the amount of rotation and scale experienced by the object.

The database of images consists of 12 objects: four keys, four mechanical parts, and four leaves. The objects were chosen for their essentially two-dimensional structure. Each object was photographed using a digitizing video camera against a dark background. We emphasize that all of the images used in creating and testing the recognition system were taken at different times using various camera rotations and distances. The images are digitized to 256 × 256, 8-bit quantized pixels, and each object covers an area of about 40 × 40 pixels. This small object size relative to the background is necessary due to the nonlinear sampling of the complex-log mapping. The objects were centered within the frame by hand. This is the source of much of the noise and could have been done automatically using the object's center of mass or some other criteria determined by the task. The orientation of each memorized object was arbitrarily chosen such that their major axis was vertical. The two-dimensional images that are the output from the invariant representation subsystem are scanned horizontally to form the vectors for memorization.

Fig. 2 displays the significant improvement observed when the input is highly corrupted by noise. Specifically, the recall operation recovers a key embedded in heavy noise (SNR = −3 dB) and the recall's SNR is equal to 8.2 dB.

Fig. 3 is an example of occlusion. The unknown object in this case is an "S" like part which is larger and slightly tilted from the memorized "S." A portion of the bottom curve was occluded. The resulting reconstruction is very noisy but has filled in the missing part of the bottom curve. The noisy recall is reflected in both the SNR and the interplay between the memorized objects as shown by the histogram. The histogram shows the interplay between the memorized image and the unknown input. The "4" on the bargraph indicates that the correct class the input belongs to has been indeed identified.

Fig. 4 displays the result of locally setting a fraction of the memory

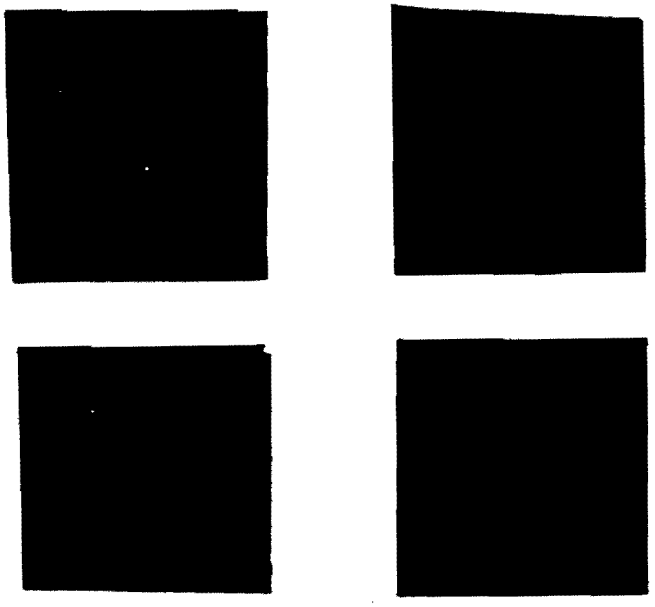


Fig. 2. Recall improves the SNR.

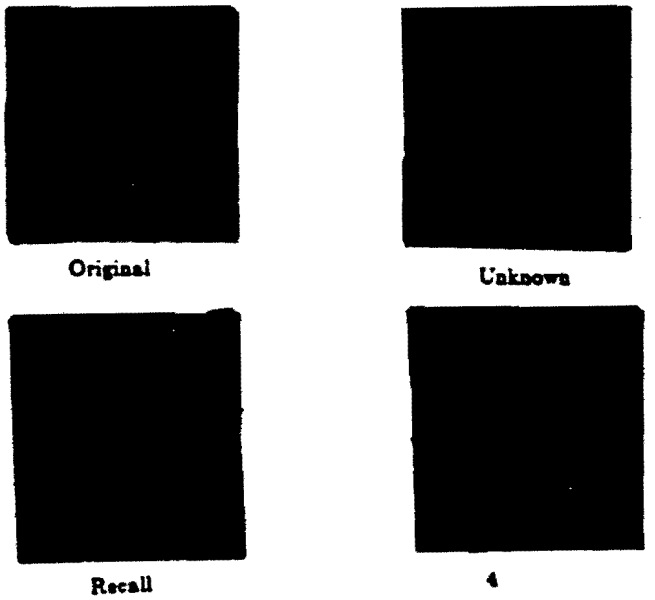


Fig. 3. Recall using scaled and rotated "S" with occlusion.

matrix elements to zero. The damage is done locally and it is present in a local sense in the recall. The warping to the complex-log mapping makes the objects of the size that were memorized tolerant to local faults in the memory. (For invariance reasons the input images were complex-logged mapped before the memory matrix is built.) The upper left image is the ideal reconstructed recall with no damage to the memory matrix. The upper right image is the recall when 30 percent of the memory is set to zero. The lower image is the recall for 50 percent local damage.

Fig. 5 is the result of randomly setting the elements of the memory to zero. The effect of this kind of damage is not nearly as critical as in the case of the local damage. The upper left image is the ideal recall. The upper right image is the recall after 30 percent of the memory has been set to zero. The lower left and right recalls correspond to 50 and 75 percent random damage, respectively. Even when 90 percent of the memory matrix has been set to zero (not shown in the figure) a faint outline of the pin could be seen in the recall. This is evidence that not all the connections in this network are necessary and as a consequence one could and should look for finding a data compression scheme for the memory matrix.

IV. Database Retrieval

We have developed and implemented the DAM-based model for fault-tolerant information retrieval. Efficient index (key) retrieval capability is essential for querying large databases [3], [24]. Furthermore, our model allows us to retrieve information in the presence of noise (errors) in the input key (index) and/or memory itself. Our results are outlined below.

A. DAM Model for Database Retrieval

We created a database of names from a student directory. The DAM model thus appears like an autoassociative memory where prestored names can be recalled using noisy inputs. There are two phases in the operation of the standard autoassociative DAM as described in Section II.

During the memory construction phase every name is encoded by a hash function (see Section IV-B) into a stimulus vector of a fixed length. Let N be the name matrix with l rows and k columns where k is the number of names in the database and l is the maximum size of any name in the database. Let S be a matrix with m rows and k columns. Each column of S corresponds to the hash vector (dimension m) of one name in the database. Instead of evaluating the memory matrix NS^+ during the database creation phase (as discussed in Section II), we only determine the generalized inverse (S^+), and store both N and S^+ for use during retrieval. During the recall phase, the input key is converted to its hash vector 't'. We multiply S^+ by 't' to obtain a vector 'c'. This vector 'c' is treated like a histogram that indicates how close the vector 't' is to each of the stored hash vectors. If $c[i]$ is the maximum element in vector 'c' then the ith hash vector is closest to 't'. The system, therefore, returns the ith name in the name matrix N as the output of the recall phase. In situations where more than one element in 'c' is very close in value (e.g., 1 percent) to the maximum element, the system makes a note of all those elements and returns the names corresponding to all those elements. Note that in our DAM model for retrieval the recalled name is one of the prestored names "closest" to the noisy input whereas in image recognition applications, the recalled image is usually a weighted linear combination of prestored images.

B. Hashing Methods

A hashing function used to encode names should satisfy the following requirements.

1) The DAM model expects all stored vectors to be of the same size. Therefore, a name to be stored in the database is transformed into a real-valued vector of fixed length using some hashing function. This fixed length vector then serves as a stimulus to the model.

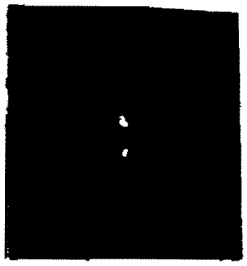
2) The hashing function must have the property that a typical error (e.g., mispelled/omitted/additional letter) in the input key at the time of retrieval will produce a hash vector that is close to the hash vector corresponding to an error-free name. This implies, for example, that the standard ASCII encoding for letters would not work because an insertion error (additional letter) or deletion error (missing letter) in the input key will result in a stimulus vector totally different from the stimulus of an error-free name. We worked with three kinds of hash functions known as: *n-gram extraction methods* ($1 \leqslant n \leqslant 3$).

The hash function creates a vector that is $26^{**}n$ elements long. Each element represents an n-gram corresponding to a set of n adjacent letters. An element has the value one if the n gram to which it corresponds appears at least once in the name. For every n gram that is absent in the name, the corresponding element in the vector is set to zero. We shall also refer to these methods as alphabet extraction, bigram extraction, or trigram extraction, depending on the value of parameter n.

Using the frequency of occurrence of n-grams trigrams in the English language should improve the performance of this model. We, however, obtained very good results even without the use of frequencies.

C. Experimental Results

The algorithms for the model discussed in Section IV-A were implemented in a "Unix" environment on a Sun 3/75 workstation.



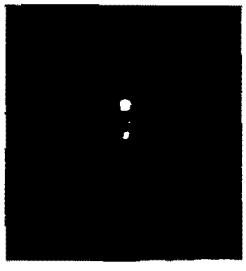
Ideal Recall

30% of Memory Set to Zero

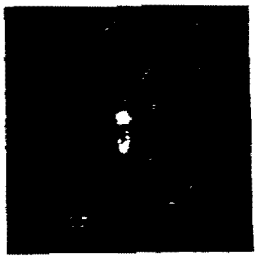
50% of Memory Set to Zero

Fig. 4. Recall for memory matrix locally set to zero.

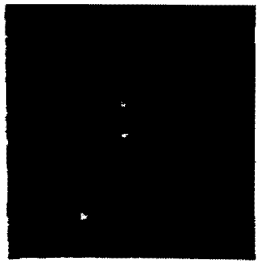
Ideal Recall



30% of Memory Set to Zero



50% of Memory Set to Zero



75% of Memory Set to Zero

Fig. 5. Recall for memory matrix randomly set to zero.

A small database was created with names taken from a student directory. Each name consisted of up to 11 letters and we considered the following two types of data:

- random (noncorrelated) data, i.e., names were extracted randomly from the student directory
- highly correlated data, i.e., names were taken from the same page of the student directory.

Fault-tolerant retrieval is analyzed with respect to the following common types of input errors:

- Single letter deletion errors;
- single letter substitution errors.

As a measure of fault-tolerant retrieval, we consider the percentage of correct retrievals with respect to a particular type of error in the input key. In our experiments, we consider the following factors which affect fault-tolerant retrieval:

- type of data stored, i.e., random versus correlated;
- database size, or the number of names stored;
- type of hashing function used (alphabet, bigram, trigram extraction);
- type of input errors.

Results are presented in the form of tables showing the percentage of correct retrievals as a function of database size. Fig. 6 shows the performance of fault-tolerant retrieval for random data. As expected, the performance degrades with increase in database size for all three hashing methods. The results obtained with the alphabet extraction method [Fig. 6(a)] also exhibit the same trend except for a quirk at a database size of 28 that we have not been able to explain. For correlated data, experimental results obtained with alphabet extraction and bigram extraction are slightly worse than for random data. However, the trigram method yields perfect retrieval even with

Words in Memory	% Correct Retrievals
8	100
12	92.4
16	82.9
20	67.7
24	50.84
28	56.7

Substitution Errors

Words in Memory	% Correct Retrievals
8	100
12	97.16
16	96.7
20	83.0
24	70.75
28	82.3

Deletion Errors

(a)

Words in Memory	% Correct Retrievals
8	100
16	100
20	100
24	100
28	91.5
70	90
100	87.5
200	69
300	14.5

Substitution Errors

Words in Memory	% Correct Retrievals
8	100
16	100
20	100
24	100
28	98
70	96
100	92.5
200	70
300	15

Deletion Errors

(b)

Words in Memory	% Correct Retrievals
8	100
16	100
20	100
24	100
28	100
70	100
100	100
200	100
300	100

Substitution Errors

Words in Memory	% Correct Retrievals
8	100
16	100
20	100
24	100
28	100
70	100
100	100
200	100
300	100

Deletion Errors

(c)

Fig. 6. Fault-tolerant retrieval for random data. (a) Performance of the alphabet extraction method on random data. (b) Performance of the bigram method on random data. (c) Performance of the trigram method on random data.

correlated data. Fig. 7 shows the elements of the recalled histogram vector '*c*' obtained when recalling a name from a correlated database (bigram method) of size 8. As expected, the trigram method shows better fault tolerance than the bigram method which in turn shows better results than a simpler method based on alphabet extraction. Although we did not gather any statistics about other kinds of errors

Hashing Method : Bigram Method
Size Of Database : 8
Type Of Data : Correlated
List Of Names : cesn
cesnanes
cesnenka
cesnik
csaoot
csace
csadbourn
csae

Input Key : cesnik

1) Input Key : cesnik
Contents Of Histogram Vector 'c' [0. 0.0.1.0.0.0.0.0]

2) Input Key : cesnik
Contents Of Histogram Vector 'c' [.15. .1..01. 0.98. 0. .02..0.0.1]

Fig. 7. Contents of the recalled histogram vector '*c*'.

% Correct Retrievals	Alphabet Extraction	Bigram Extraction	Trigram Extraction
100%	4	20	—
66%	20	200	—

Fig. 8. Saturation effects observed on correlated data.

(e.g., insertion errors, multiple letter deletion and substitution), we found that the system handles them very well.

Our experimental results can be explained qualitatively using the general DAM analysis. Namely, according to [23] when a noisy version of a memorized input vector is applied to the memory, the recall is improved by a factor corresponding to the ratio of the dimension of the vectors to the ratio of the number of memorized vectors.

We performed many experiments to analyze how robust retrieval is affected by the size of memory (DAM saturation effect). Fig. 8 shows the maximum database size for which the percentage of correct retrievals (assuming deletion errors) exceeds a certain value. The table shows that when the database size exceeds 20, the alphabet extraction method's retrieval rate falls below 66 percent, while that of the bigrams deteriorates below 66 percent only when the number of names stored exceeds 200. The trigram method yields good results even when more than 300 names are stored. This illustrates the saturation effect typical of DAM-based models. We did not continue to experiment with larger databases, to determine when the trigram method will saturate, because the computation time for the trigram method is quite large. Saturation is a consequence of the crosstalk effect caused by the memory being nonorthogonal due to similarity among the stored stimuli. This explains why the alphabet extraction method with a vector size of 26 saturates faster than the bigram method with a vector size of 26^2 which in turn saturates faster than the trigram method with a vector size of 26^3.

The absolute upper bound on the number of vectors that can be stored in the database is the dimension of the hash vector. In practice, however, this limit is never achieved because one does not have control over the orthogonality of hash vectors of prestored names. Moreover, some of the names must have hash vectors that are linearly dependent on the previous vectors and thus prevent retrieval of that name altogether. This never happened with the trigram method, it happened for one of the 300 correlated names stored in the bigram method and for four of the 26 correlated names stored in the alphabet extraction method. This rare condition is detected during memory construction.

The DAM database is a distributed memory. Therefore, when a portion of the memory is corrupted, the system does not lose its

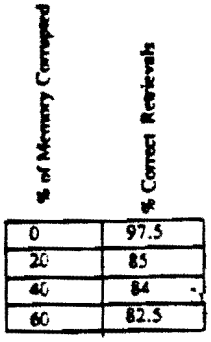


Fig. 9. Graceful degradation in the case of memory corruption.

retrieval capabilities altogether but degrades gracefully. Portions of the memory were randomly selected and set to 0.5. The results are shown in Fig. 9. Initially, we had set portions of the memory to zero, but this did not seem to affect the performance of the system at all because the original (error-free) memory is sparse. This experiment was performed on a database of 28 correlated names that used the bigram hashing method. Overall we have shown that the DAM database is very resilient to corruption in the memory itself.

V. Hardware and Software Implementations

The computationally intensive part of the algorithm for computing the generalized inverse of a matrix [20] is the Gram-Schmidt orthogonalization process, or matrix triangularization. Triangularization on a sequential computer is computationally expensive since it requires $O(n^3)$ operations for an $n \times n$ matrix. Fortunately, one can use systolic arrays for the implementation of matrix operations in VLSI. At least two systolic array designs have been proposed to perform triangularization for general (dense) matrices [1], [6]. Both designs use triangular systolic array and are based on Givens rotations to guarantee numerical convergence and stability. The difference essentially being that in [6] the input matrix is entered one row per step and in [1] it is entered one-column per step. Matrices that are too large for a given systolic array can be triangularized by first splitting them into blocks [1]; in this case, the triangularization would require $O(n^3/p^2)$ time where n is the matrix dimensionality and p^2 is the number of processors in systolic array, $p < n$.

Matrix-vector multiplication for the recall operation and/or the gradient descent learning technique can be also implemented using VLSI systolic arrays [12]. Since the matrix M is very large, we cannot expect to fabricate a systolic array of a corresponding (large) size on a monolithic chip due to I/O bandwidth limitations. An approach based on partitioned matrix algorithms [13] has been proposed to overcome this problem by using smaller VLSI array modules which perform computations on $p \times p$ submatrices and $p \times 1$ subvectors.

We are currently trying to implement the DAM on an NCUBE parallel computer. Since the DAM model involves large-scale matrix operations, we can achieve significant speedup using a parallel NCUBE machine where a large matrix is partitioned into several block submatrices, which are stored and manipulated in different nodes of the hypercube.

VI. Conclusions

We describe in this paper the distributed associative memory (DAM) model for distributed and fault-tolerant computation as related to image recognition and database retrieval tasks. Our experiments, as reported in this paper, show the feasibility of our approach and prove the strong connection between pattern recognition (cluster analysis) and database research.

The fault-tolerant aspect relates to the robust performance DAM's exhibit in the presence of noisy inputs and/or memory faults. Experiments which included both local and global faults in the memory showed that the retrieval is still acceptable. We are presently concerned with the possibility of regenerating a faulty memory to its original condition and with developing alternative hashing methods for improving on the capacity of the memory (i.e., to decrease the saturation effect).

References

[1] R. P. B. Bojanczyk, R. P. Brent, and H. T. Kung, "Numerically stable solution of dense system of linear equations using mesh-connected processors," *SIAM J. Sci. Stat. Comput.*, vol. 5, no. 1, pp. 95–104, 1984.

[2] H. J. Caulfield and M. H. Weinberg "Computer recognition of 2-D patterns using generalized matched filters," *Appl. Opt.*, vol. 21, no. 9, 1982.

[3] C. J. Date, *An Introduction to Database Systems*. Reading, MA. Addison-Wesley, 1986.

[4] S. E. Fahlman and G. E. Hinton "Connectionist architecture for artificial intelligence," *Computer*, pp. 100–109, 1987.

[5] J. Feldman, Ed., *Cognitive Sci.*, vol. 9, Special Issue on Connectionist Models and Their Applications, 1985.

[6] W. M. Gentleman and H. T. Kung "Matrix triangularization by systolic arrays," in *Real Time Signal Processing IV, SPIE Proc.*, vol. 298, 1981, pp. 19–26.

[7] R. Hecht-Nielsen, *Artificial Neural System Technology*, TRW AI Center, 1986.

[8] C. Hester and D. Casasent, "Interclass discrimination using synthetic discriminant functions (SDF)," in *Proc. SPIE Infrared Technol. Target Detection Classification*, vol. 302, 1981.

[9] W. D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.

[10] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci. USA*, vol. 79, April 1982.

[11] D. Hubel and T. Wiesel "Brain mechanisms of vision," *Scientif. Amer.*, Oct. 1979.

[12] K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.

[13] K. Hwang and Y.-H. Cheng, "Partitioned matrix algorithms for VLSI arithmetic systems," *IEEE Trans. Comput.*, C-31, pp. 1215–1224, 1982.

[14] T. Kohonen, *Self-Organization and Associative-Memories*. Berlin, Germany: Springer-Verlag, 1984.

[15] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP*, pp. 4–22, 1987.

[16] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, Eds., *Parallel Distributed Processing*, vol. 1, 2. Cambridge, MA: MIT Press, 1986.

[17] D. Marr, *Vision*. San Francisco, CA: W. H. Freeman, 1982.

[18] M. Minsky and S. Papert, *Perceptions*. Cambridge, MA: MIT Press, 1969.

[19] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan, 1962.

[20] B. Rust, W. R. Burrus, and C. Schneeberger, "A simple algorithm for computing the generalized inverse of a matrix," *Commun. Ass. Comput. Mach.*, pp. 381–387, 1966.

[21] T. J. Sejnowski and C. R. Rosenberg, "NETtalk: A parallel network that learns to read aloud," EE and CS Dep., Johns Hopkins Univ. Tech. Rep. JHU/EECS-86/01, 1986.

[22] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Commun. Ass. Comput. Mach.*, vol. 29, pp. 1213–1239.

[23] G. S. Stiles and D. L. Denq, "On the effect of noise in the Moore-Penrose generalized inverse associative memory," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 7, no. 3 pp. 358–360.

[24] H. S. Stone, "Parallel querying of large databases: A case study," *Computer*, Oct. 11–12, 1987.

[25] H. Wechsler and L. Zimmerman "2-D invariant object recognition using distributed associative memory," *IEEE Trans. Pattern Anal. Mach. Intell.*, 1988, to be published.