Paired Associative Memory:
A Simplified Semantic Network LTM Model and Distributed "Neural" Implementation

Ted Selker

COINS Technical Report 81-20

(September 1981)

This work done in partial fulfillment of Masters of Science Degree January, 1981.

PAM:

PAIRED ASSOCIATIVE MEMORY

A model of organism memory

Ted Selker

Masters Degree Project

Department of Computer and Information Science

University of Massachusetts at Amherst

January 1981

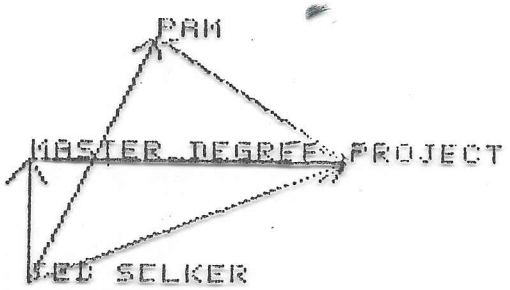
Abstract

Paired Associative Memory (PAM) is presented as a model of organism Long Term Memory. It is composed of association pairs. Either associated element in a pair can be representations of another pair or stimulus "terminal". The model is described in two languages; the network model with its nodes and arcs is related to a "neural" layered distributed model.

Requirements for a model of Long Term Memory are developed through a discussion of literature relating to physiological and psychological data and models for organism memory. Examples of progress of Artificial Intelligence (AI), Psychology and Brain Theory memory models are reviewed.

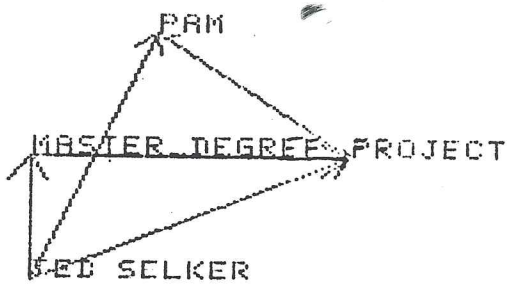
PAM is first described as a network style model to clarify some of the important features of the distributed model. In a simulation on a VAX11/780in UMASS LISP with color graphics, PAM is shown to be able to encode knowledge in hierarchical structures, some parts of which are called schemas. It can encode new data in terms of established knowledge. To the extent that it creates such hierarchies, we say it bootstraps itself. As well, it has the property that it can encode knowledge in unfamiliar domains without having to be modified.

fig. 1



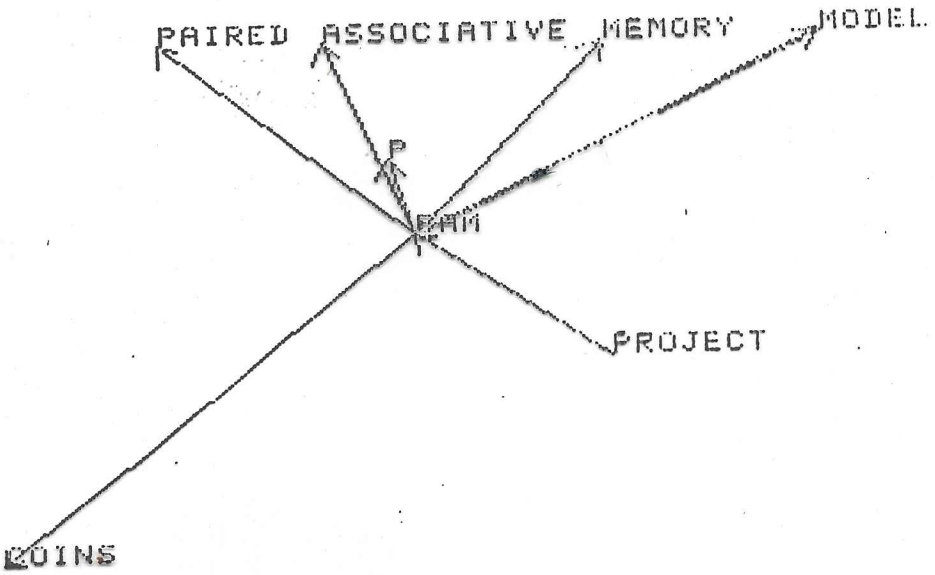

```
FUNCTION:    EKE-RULE
YOU ARE AT: PAM.
```

fig. 1



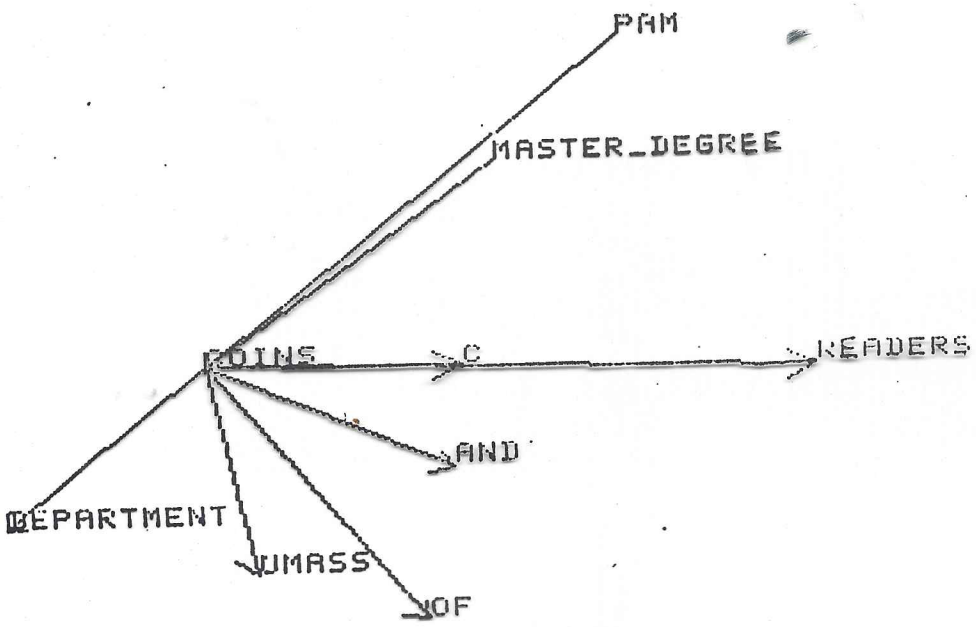
FUNCTION: EKE←RULE
YOU ARE AT: PAM.

fig. 2



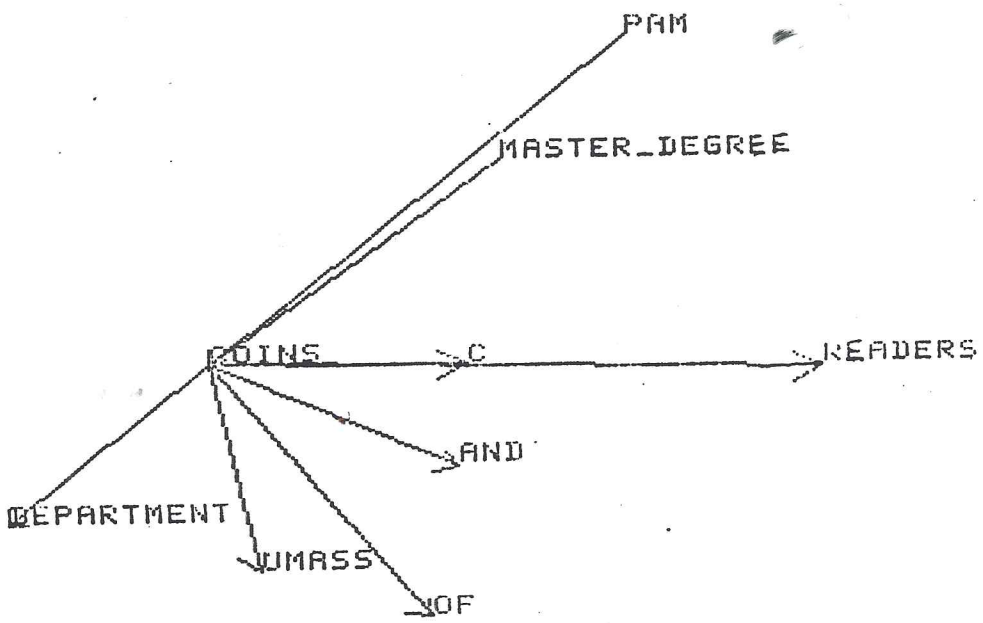

```
FUNCTION:   FROMHERE
YOU ARE AT: PAM
```

fig. 3



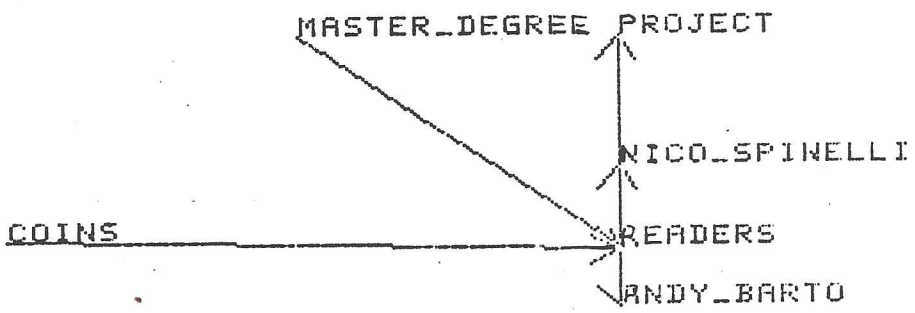
FUNCTION: FROMHERE
YOU ARE AT: COINS

fig. 3



FUNCTION: FROMHERE
YOU ARE AT: COINS

fig. 4



FUNCTION: FROMHERE
YOU ARE AT: READERS

fig. 5

PAIRED ASSOCIATIVE MEMORY MODEL

P A M

PAM

MASTER_DEGREE PROJECT

TED SELKER NICO_SPINELLI

COINS C O I N S READERS

ANDY_BARTO

AND

DEPARTMENT COMPUTER INFORMATION SCIENCE

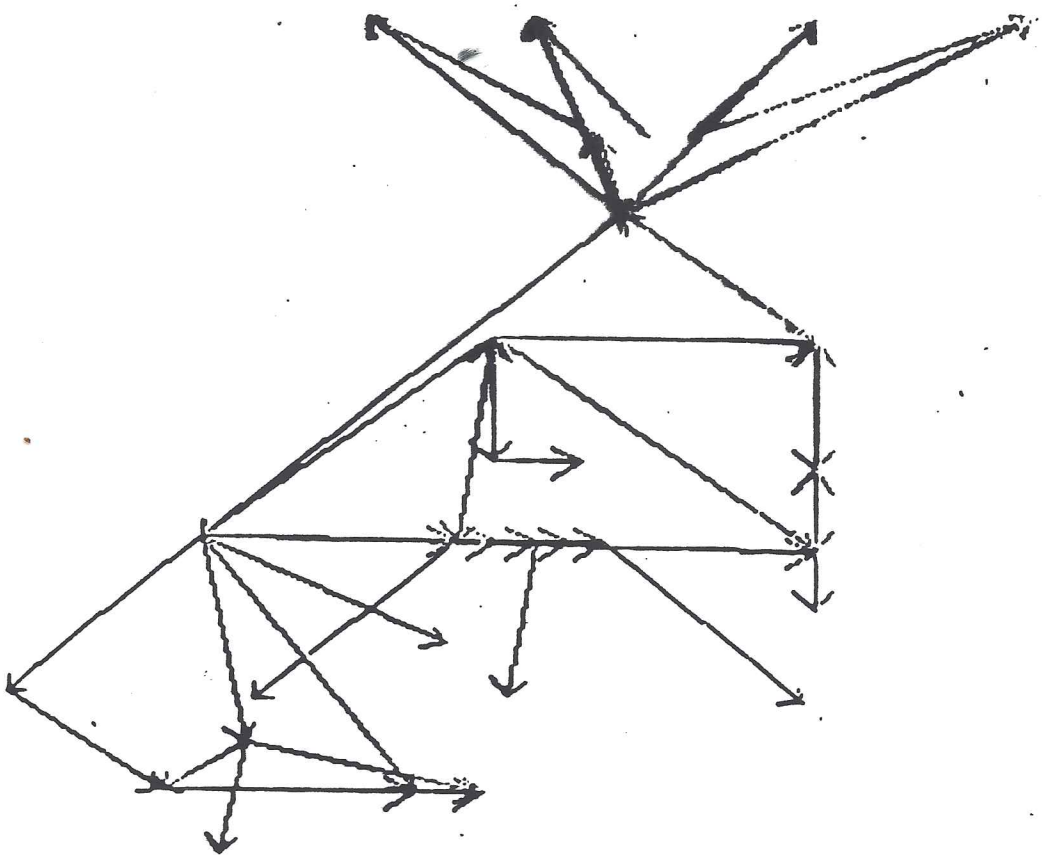
UMASS

UNIVERSITY OF MASSACHUSETTS

AMHERST

FUNCTION: DRAW-WORLD
YOU ARE AT: READERS

fig. 6



FUNCTION: CMNDO
YOU ARE AT: RERDERS

Table of Contents

Table of Figures

Figures appear following pages

Acknowledgements

Many people helped me with this, my Masters Degree Project. First I would like to thank Dr. Nico Spinelli for the provocative questions which inspired me to propose a memory model. His criticism has guided the project.

I would like to thank Dr. Andy Barto for insisting that my simulations be as simple as possible and that they be performable and clear. As well, he has given me constant encouragement to complete the project.

I am deeply indebted to my dear friend Marietta Storm for staying up nights drawing figures and helping me go over and over syntax and spelling.

I would like to thank my friend Anne Paulson who helped me through some of the most tiring work by locking me in various rooms and forcing me to work.

I am forever thankful to Lisa and Zachary Smith for helping me think some of the fundamentals of rewriting.

My fellow graduate students helped me as well: I appreciate the hours which Rich Sutton gave freely to help me question my ideas. He lead me to related material and share pertinent software. I, as so many other COINS graduate students, am indebted to Terry Weymouth and Frank Glazer for their warm friendly manner and help with system software.

Many other people helped me to finish my Masters Degree Project. I would like to thank people who read parts of the paper and helped me with spelling. I must thank my housemates who showed so much patients and encouragement. Finally, I must thank my family whose long distance support I appreciated.

Introduction

> "There is a sense in which it can be said that the methods of science have scarcely yet been applied to human behavior."
> (Skinner, B.F. 1971).

Although many of Skinner's intuitive ideas are controversial at best, the above quotation represents one instance where his beliefs coincide with those of many others. In fact, many people would expand the meaning of the words "human behavior" in this quotation to include the behavior of all developed organisms. In this sense, Developed Organism Behavior, would denote problem solving abilities and conscious problem modeling as well as language. Accordingly, developed organized behavior may be analysed by the application of scientific methods. The crucial question then becomes: what methods will forward the understanding of such behavior? Recently, scientists have been able to examine such problems as self-defined behavior and problem solving by turning their attention to high level concepts of problem solving such as hypothesis forming and testing. Successful experimentation using modeling is beginning to clarify the dynamics of behavior and establish the usefulness of modeling as a scientific method.

A good Leaning Tower of Pisa and some cannon balls are adequate tools for testing a model of gravity. Models of aspects of Developed Organsim Behavior are more difficult to

evaluate. Tremendous effort in several fields has generated exciting perceptual, neurophysiological, and neuroanatomical data about organisms. Even data from all these experiments and their associated technology do not by themselves seem to be adequate tools for understanding developed organism behavior.

The difficulty of understanding thought is due in part to its apparent nondeterminism. This nondeterminism is exemplified by the complex and non-physical level of the inner mechanisms which organisms use to make decisions. In a computer, as in organisms, the important data about how it works is the most difficult to obtain. What are the brain's "instruction set" and "software source code"? On a computer, these are exactly what the naive user, who is only allowed to interact with the text editor, does not have available to him/her for inspection. We are naive users of our brains; we only interact with its user interface. We are able to learn and remember, without knowing what the inner physical and structural mechanisms for learning and memory are. Much is known about the anatomy and physiology of the brain and about perception, and psychological characteristics of learning and memory. Hypotheses must be made and tested to fit all of these elements together. Modeling and simulation could be the "scientific tools" which will be most useful for testing hypotheses about memory learning and

associated organism behavior.

"...Almost everyone attributes human behavior to intentions, purposes, aims and goals." (Skinner, B.F. 1971)

Although many workers in Artificial Intelligence (AI) would not support most of Skinner's viewpoints, almost none of them would deny the usefulness of separating developed behavior into "intentions, purposes, aims, and goals." Most modern researchers in AI give explicit credit to problem reduction as a method of problem solving (Nilsson, N.J. 1971). Some landmark models of problem representation boast the use of this intuitive and powerful technique, which represents problem and solution spaces in terms of a resolved adaptive protocol which builds its own heuristically based solution space. Such ambitious projects as General Problem Solver (GPS) (Newell, A.; Shaw, J.C.; and Simon, H.A.; 1960) and Hearsay (Erman, L.D.; Lesser, V.R.; 1975) are based on this philosophy.

Problem reduction, breaking solutions into intentions, aims and goals using hierarchical representations including goals-subgoal and actions is not as new an idea as it is powerful. It has been applied to large systems and is implicit in control hierarchies in government and military command structures, business management organizations and industrial manufacturing procedures. For over one hundred years, it has been assumed to be a mechanism used in the brain for generating and controlling behavior (Jackson, J.;

1931).

What Is An Appropriate Scope For Present Day Brain models?

To assume that any one mechanism in a model can be proposed for the mechanism or structure of the whole brain is to deny the anatomical structure of the brain. The brain is a collection of interacting processing and perceptual interfaces which have select and overlapping functions. The heuristics for control in an organism with lesions in its hindbrain are, and should be expected to be, very different than those which control an organism which is using a brain which is missing a frontal lobe, parts of its visual tract or its cerebellum (Carpenter, M. 1969). Each of these are different organisms which have predictable and distinct behavioral anomalies. These characteristic, or stereotyped deficiencies are evidence of specificity of function and control for specific parts of the brain.

This is not to deny the results of Lashley's mass action experiments (Lashley, K.S. 1929). In these experiments, he found that animals were able to perform with an ability inversely proportional to the mass of cortex which he ablated from the organism. Whether through redundant or distributed function, the brain is designed to gracefully degrade in performance as parts of it become nonfunctional. The heuristic nature of the way higher animals plan and solve problems allows their solutions to problems to be

based on any one of a number of perceptual and cognitive resources. When some of these resources are destroyed, the brain relies on others. Contrary to the letter of Lashley's results, close analysis of a brain-damaged organism shows specific deficiencies. People with temporal lobe seizures, for example have specific stereotyped personality disorders (Sherwin, I. 1976). The severity and characteristics of specific personality characteristics is directly related to the developmental time period in which the seizure began as well as the size of the seizure focus.

The perspective of multiple and different loci of control, with distributed as well as redundant memory structures, seems crucial to the stability of the brain. Any model which proposes to describe all brain functions would have to account for each structure in the brain. Therefore, a detailed accounting of all aspects of the brain's behavior should be attempted only when models of individual brain functions have been accepted. Consequently, although this paper describes a hierarchical modeling language for memory and learning, and proposes a neural model in which it could be described, it does not model the modulation of learning and retrieval abilities or the function of the reticular formation and other brain control structures. We attempt to motivate the model somewhat by describing other people's modeling efforts.

Taxonomies of Memory and Learning Models

Hebb (Hebb, D.O. 1949) describes frustration with people's attempts to model behavior using models which do not try to account for the time between stimulus and response as a process. He identified two classifications. Switchboard theory accounted for models in which input was wired to output directly. Field theory used concepts of physics field theory to describe distribution of sensory excitation and ratios of excitation as causing motor responses. Happily, the thrust of all but Skinner's present day theories speak directly to this intervening thought part of behavior. These theories of thought and memory can still be categorized as having simple switchboard or distributed field theory memory connectivity patterns.

Modeling of Long Term Memory (LTM) has recently become one of the focal points of workers in many fields. The effective accessibility of LTM, it appears, can account for much of developed organism behavior. Psychologists have categorized their models as network, set, or semantic feature models (Klatzky, R.L. 1975).

So-called network models include models of memory in which the memory is described as a vast network of associations. Set-theoretic models are those which represent

knowledge in semantic categories. People in Artificial Intelligence have used the term "Frames" (Minsky, M. 1976) to describe knowlege sources which perfom actions in environments which can be characterized by categories. The separation of information by organism's brains into interrelated bundles is empirically supported as well as being intuitively pleasing. The concept of a category or schema is an old one, central to Hebb's book (Hebb, D.O. 1948). models with this theme have been numerous and have been put forward with emphasis on being interpretable, from the network viewpoint (Marr, D. 1970; Maclaren, L.S.; 1980, and many others), as well as the semantic feature model viewpoint. a rich source of examples which make

A taxonomy of models of concept formation put forward by Richard Millward (Millward, R. 1978) attempts to further classify memory models into Prototype, Exemplar, Frequency and Rule models.

Prototype models include production rule models (Rychener 1976) in which a list of rules is available and can be enacted when applicable (and hopefully extended when necessary). These prototypes can be semantic or spatial. It has been shown that prototypes are used extensively by people for encoding memory. In a famous study (Posner and Keele 1968), subjects were found to be able to remember new

dot patterns, which were related to a prototype, much better than other equally unfamiliar dot patterns. Such a phenomenon has recently become famous and documented for Master Chess players' ability to remember large numbers of chess pieces positions when arranged in classical chess game patterns, but not many positions for chess pieces arranged randomly on a chess board (Larkin, J.; McDermott, J.; Simon, D.P.; and Simon, H.A.; 1980).

Exemplar models are models which can be described in terms of cluster analysis. An overview of some of the techniques of cluster analysis is available (Duda, R.O., and Hart, P.E.; 1973). The idea is that memories are categorized according to distance from the center of clusters. Distance is measured by a multidimensional classification of sameness of some sort. Although Millward states that he believes this to be a counter-intuitive notion and tries to point to evidence to this effect, the concept of a similarity measure has been an important feature of many modern models (Rips, L.J.; Shoben, E.J.; and Smith, E.E.; 1973; Marr, D.O. 1970; Nakano, K. 1972 et al). The tip-of-the-tongue experiments (Brown, R.W., and McNeil, D.; 1966) show that, in fact, when trying to retrieve things from memory, generic information is available.

In their study, Brown and McNeil found that people try-

ing to recall things in their brains search for items which are similar (sounds like, looks like, tastes like, or starts with some clearly retrievable semantic entity). Relatedness between memories affects other aspects of recall performance as well. Until recently, Psychologist's experiments portrayed human recall as better than recognition performance in all situations; however, in a landmark demonstration it has been shown that performance for recognition can be better than recall (Thomson, D.M., and Tulving, T.E.; 1970) when highly similar things in memory interfere with recall. Such experiments support the notion that memories are categorized according to distance in a multidimensional classification of sameness of some sort.

Frequency models categorize knowledge by incrementing counters for preset categories. In this way, new information is supposed to be categorized in terms of old clusters.

Rule models claim that we learn by hypothesis testing and rule forming. These rules may be memories in other models above. Rules don't govern all learning. In fact, young children go through at least 3 stages of learning. Until they are 3 or 4 years old, they are unable to learn rules. Between the ages of 3 or 4 and 6, they are unable to break rules and after that they can accept and generate exceptions to rules (Piaget, J. 1963).

Any model which has as its goal explaining perceptual data for memory has to be able to represent rules and exceptions to rules, schemas, etc. It should be able to be represent new knowledge in terms of stable knowledge, thereby allowing a hierarchy of knowledge. The model should be formulated so that a key or part of a stimulus can elicit the whole stimulus. An example of this is how a few bars of a song are enough to elicit the whole song. To be as reliable as it is, memory needs to be structured in a distributed way, and/or have an enormous amount of redundant encoding. This reliability is what gives learning and memory in organisms a graceful degradation in performance in the event of missing cells, and/or structures. In some way, the memory has to allow recall to be in terms of closeness of stimulus similarity. Furthermore, as we shall see, memory should be encoded in critical features, rather than nonsemantic criteria, such as absolute cartesian coordinate space or the alphabet.

Some Sample models of Memory and Learning

Before attempting to build a model of memory, it is instructive to examine some examples showing what has already been developed.

Hebb's Landmark model

Hebb's "cell assembly" model (Hebb, D.O. 1949) includes several noteworthy and important points. He assumes that neural cells within a small area, within area 17-20 of the brain's visual cortex, can be connected to each other to form a completely connected network if necessary. He then proposes that specific cells are connected serially to form loops of excitation. He calls these "cell assemblies". The connections between them are learned using the now famous, but largely unproven, "Hebbian Synapse". This is a synapse which tends to form when an active fiber is close to another cell's soma and the two seperate cells fire at the same time. If there is already a synapse between the two affected cells, the Hebbian rule tends to enlarge and strengthen the synapse.

In Hebb's model, cell assemblies may be sent into reverberation through direct stimulation of one of their elements. These "phase sequences" can be a succession of assembly actions. In this way, part of a "memory" can elicit all of it. This extremely important point is one of the

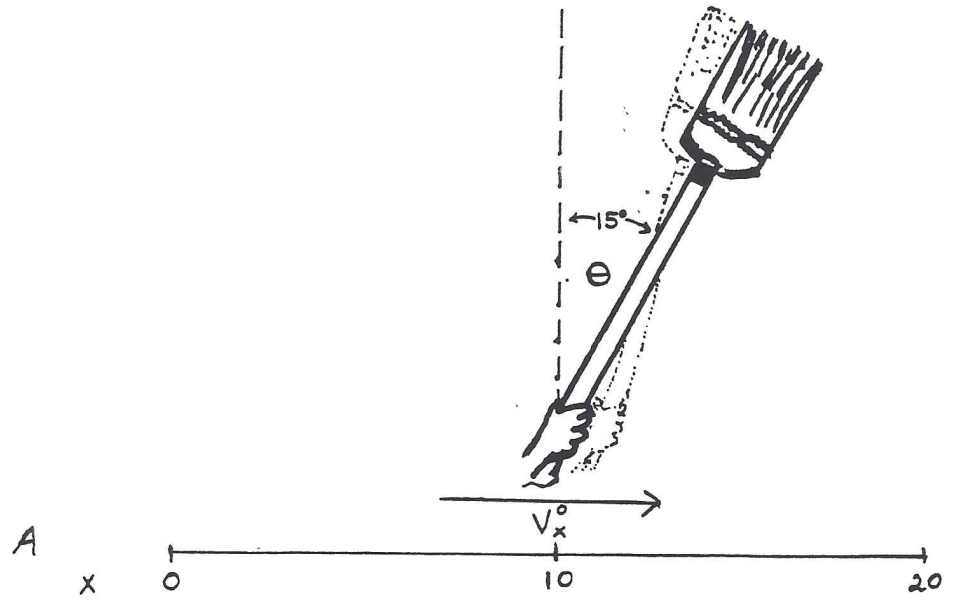
main goals of more modern models such as OCCAM (Spinelli, D.N. 1970). Hebb describes some assemblies as being based on sensory stimuli and some based on both stimuli and the state of the rest of the system. This is the point at which he could have described a hierarchical scheme in which simple cell assemblies were the basis of more complex ones. Instead, he describes an accumulation of cell assemblies interacting in vague synergy.

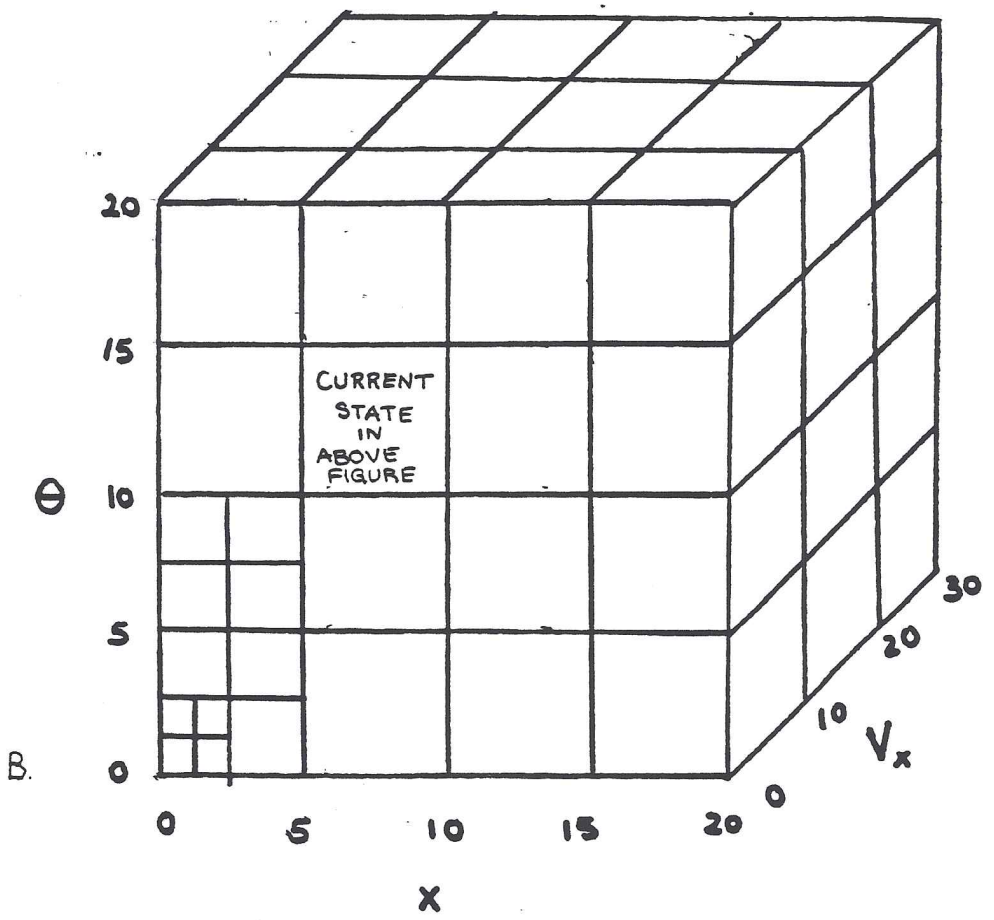
In a hierarchical scheme, "high level" "cell assemblies" could encode knowledge in terms of simple or "low level" cell assemblies. Systems which can create high level knowledge structures in terms of relationships between low level structures, we will call Bootstrapping systems. This term is an important one; it describes an ability to effectively integrate and encode data. The term bootstrapping has also been used to describe simple adaptation (Widrow, B., Gupta, N.K., and Maitra, S. 1973). This is unrelated to our definition.

Another problem with Hebb's model is that it doesn't explicitly talk about reliability. The cell assemblies are loops, serial in nature and prone to destruction, if one element becomes missing. In this way any model which relies on specific cell circuits for specific memories will be prone to destruction unless it has many circuits which have

the same memory, in different areas. Finally, his totall excitatory cell assemblies have no controlling inhibition. For this reason alone his theory cannot work without modification.

fig. 7





USING AN ASSOCIATIVE MEMORY, BOXES CAN BALANCE A BROOM HANDLE(A). BOXES REPRESENT STATES IN THE PROBLEM; HIGHLY DIVIDED BOXES (B). DISTINGUISH DECISION IN CRITICAL PART OF PROBLEM. (MICHIE, D. AND CHAMBERS, R.A. 1968

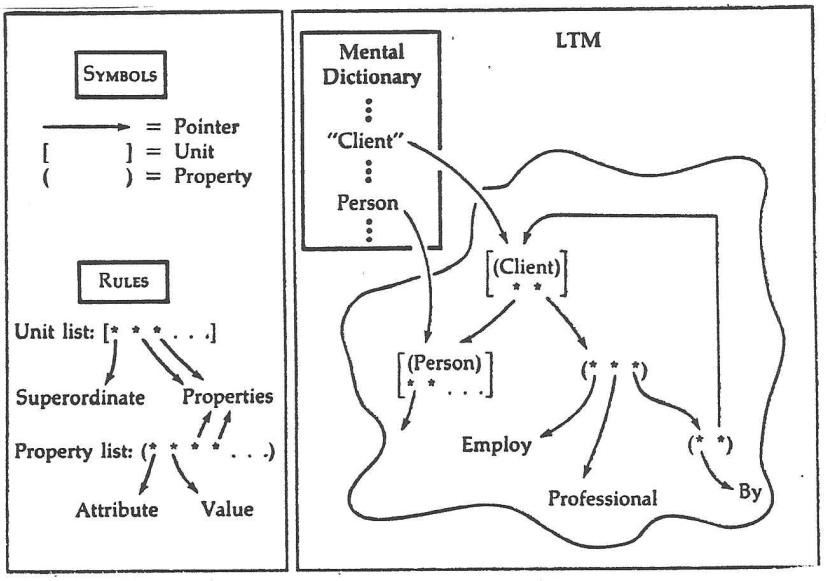
Explicit Representation of Problem Decomposition

General Problem Solver (GPS) was an AI program which developed plans, goals, subgoals and strategies to solve problems. It was developed from an earlier work, the Logic Theory Machine (LT) which was the earliest major contribution to the field of heuristic programming (Newell, A.; Shaw, J.C.; and Simon, H.A.; 1960). In this language, associative links or attributes and plans are all homogeneously representable as data (Newell, A.; and Simon, H.A.; 1963). GPS was demonstration of heuristic problem solving for Artificial Intelligence. It promises that heuristic problem solving can be simulated but does not put forward a model for how organisms store memories.

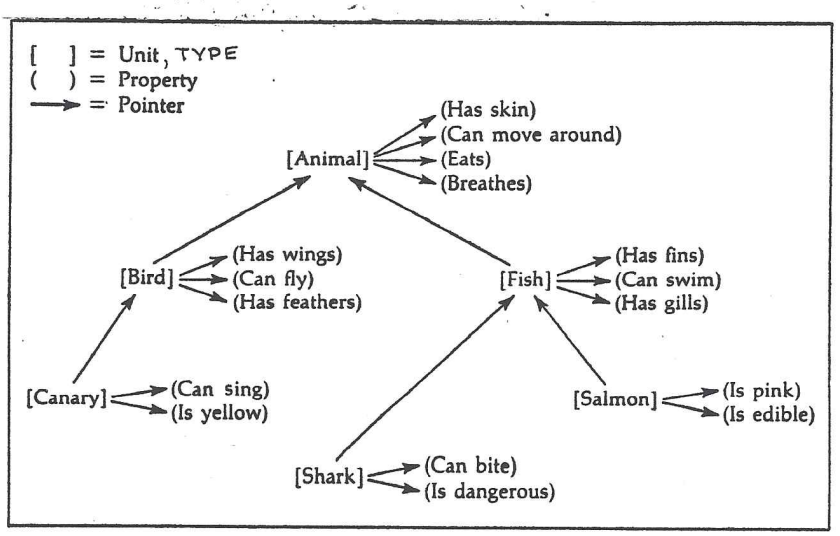
More recent, related work has lead to BK9.8 besting the world Champion Backgammon player at Backgammon (Berliner, H.J. 1980). BK9.8 is a knowledge-based program which makes decisions based on heuristics and past experience. It describes knowledge in terms of a hyperspace of features. This is an example of an associative memory. Knowledge is stored and retrieved according to its location in this feature space. Extensive work in heuristic problem solving has been done. Modern literature on it uses the idea and term Non-Monotonic Logic (McDermott, D.; and Doyle, J.; 1980).

The above examples typify AI solutions. They are generally demonstrations which solve isolated problems and are not extensible to other aspects of problems. Using this approach, programs which work on specific and narrowly defined problem domains demonstrate significant proficiency (heuristic problem solving (GPS), natural language parsing (Marcus, M. 1978), etc).

fig. 8



A. Information in TLC's memory corresponding to the concept "Client." [After Quillian, *Communications of the ACM*, Vol. 12, August 1969. Copyright 1969, Association for Computing Machinery, Inc.]

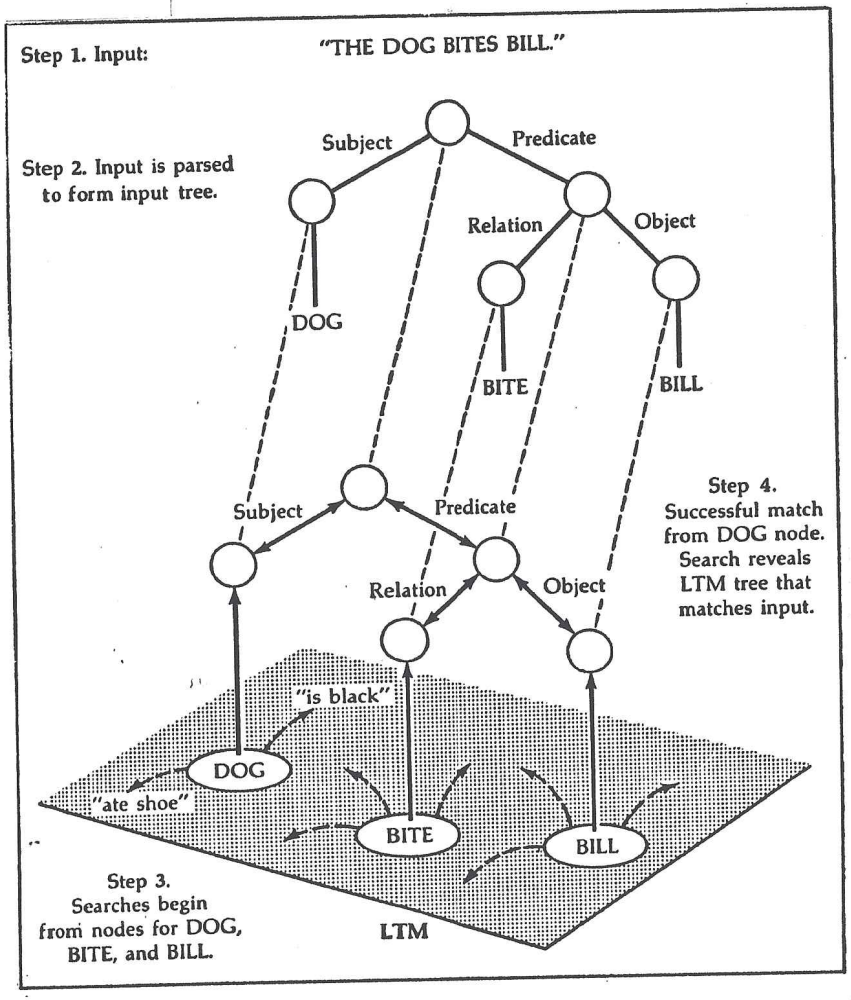


B. Portion of a hierarchy in TLC's memory, showing relationships among the units and properties within the category "Animal." [After Collins and Quillian, 1969.]

Another Associative Memory Problem Representation

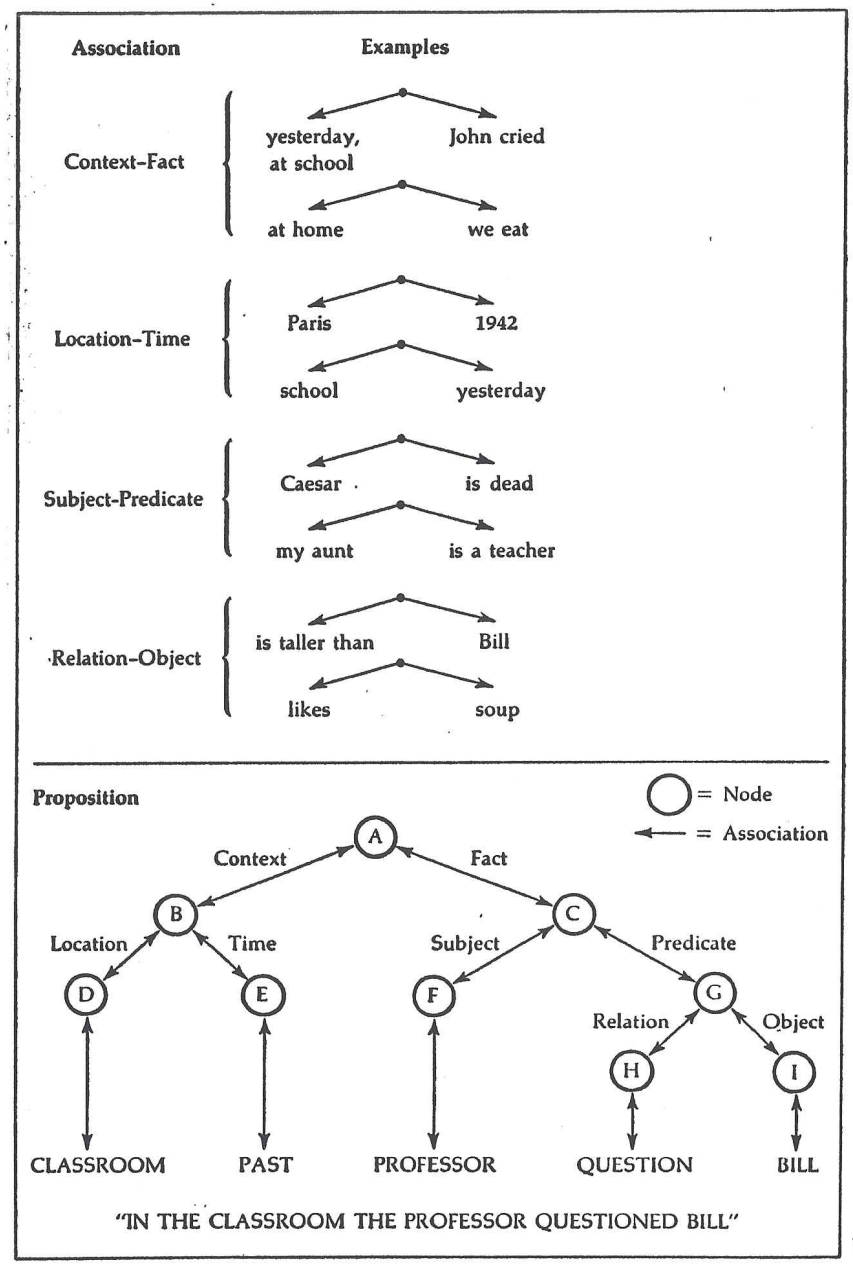
BOXES (Michie, D.; and Chambers, R.A.; 1969), like BK9.8, implements an associative memory. The concept of an associative memory is like a holograph, in that a reversible spatial transformation maps things that are to be stored onto a distributed area on the memory space. In BOXES, this transformation maps localized memory boxes onto specific areas the state space of the broom handle and cart problem (Fig. 7). BOXES presents simple memory representation with several interesting attributes. Their memory is designed to simply servo-control the balance broomstick problem, a problem which has is represented effectively in any one of a number of solution spaces. For example it can also be represented by a state space (Nilsson, N. J. 1971). Boxes in the system represent places in the control problem space which represent velocity, direction, position and broom-handle angle. The boxes have a decision making policy which relies on data from past experience for similar situations. In this way experience is somewhat generalized. In addition, the idea of dividing boxes into multiple boxes where decisions are hard to make is put forward (Fig. 7). With this addition, the model depicts more memory space for more important areas of the solution space. This is an intuitively pleasing idea that reminds one of the sensory and motor homunculi on either side of the brain's central sulcus

fig. 9



The Match process in Anderson and Bower's HAM, showing how the input sentence "The dog bites Bill" is matched with information in LTM.

A



B Types of associations and examples of each type *(above)* and a proposition tree corresponding to the proposition "In the classroom the professor questioned Bill" *(below)*, as represented in the HAM model of Anderson and Bower (1973).

(Penfield, W.; and Rasmussen, T.; 1950).

The BOXES model makes no attempt to be able to use the knowledge it acquires in new situations, such as moving the balanced upside-down broomstick to and fro at will. No effort is made to describe an expanded model which can encode and distinguish between two situations. No anatomical structure is suggested. However, in a sense the model can accomodate boxes being ablated, although specific sections of the memory space do control specific sections of the control space.

Network Models of Memory

In an effort to model memory in a simple way, some people have produced models using the arc node notation of graph theory to connect high level abstractions. Some of the most accepted theories of this type are the semantic networks (Quillian, M.R. 1969, Anderson, J.R. and Bower, G.H. 1973). Of these so-called network models, Quillian's is probably the first to described as purporting to describe human Long Term Memory. In his Teachable Language Comprehender (TLC) model Quillian has two types of nodes and a pointer. The type or unit nodes represent things. The token or property nodes describe the type node. Pointers connect nodes. Type nodes are defined by other type nodes and token nodes. In this way a complex network of definitions can be interconnected to have concept value (Fig. 8). Quillian (Quillian, M.R. 1968) explained that undifferentiated links were inadequate for representing English text. He uses subclass, superclass distinctions and conjunctive, disjunctive distinctions. In this way arcs are labeled with "And" or "Or".

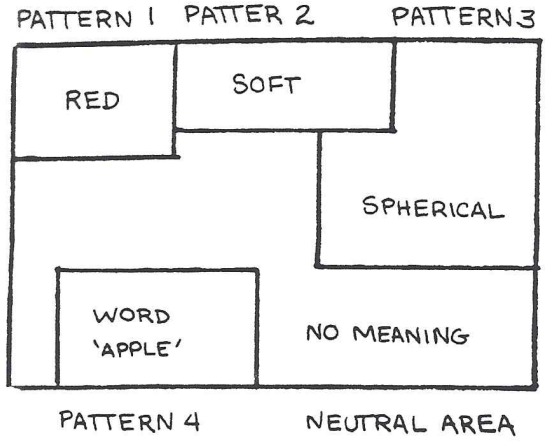
With this simple formalism, Quillian's model is supposed to be able to encode any concept. The model uses an intersection search to see if two concepts are related. If an input sentence is compatible with TLC's relations between

concepts, it is accepted. In this way, radiating, breadth first searches are used to find relationships. If the topology of the memory network is oriented on a similarity measure then the search will be improved by the topology. How the model is defined in the brain is an unapproached question. The model's formalism does not allow any way of representing the fact that associations vary in importance, and it does not attempt to give perceptual or physiological implications and implementations.

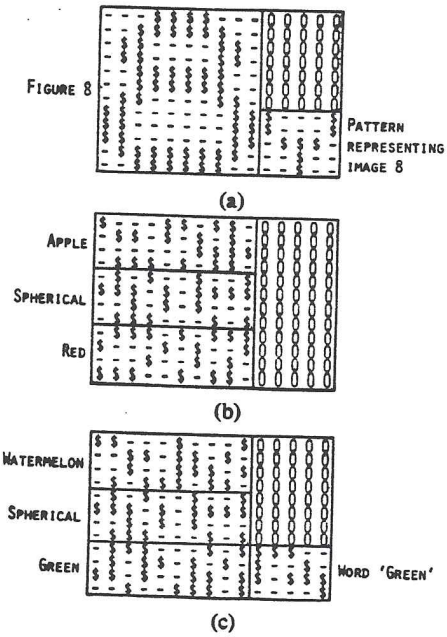
The Human Associative Memory (HAM) is a later network model of human Long Term Memory (LTM) (Anderson, J.R., and Bower, G.H. 1973)(Fig. 9). They use a linked network of nodes in which associations are more explicitly labeled. Binary associations with links make up the network. Each association in the graph is described by one of four labels. These labels are: 1) Context and a Fact, 2) Location and a Fact, 3) Subject and a Predicate, or 4) Relation and an Object. The model uses a "Match" process, to evaluate a thing's relationship to existing knowledge. In this process, sentences are comprehended if: they can be parsed (in general an unsolved problem), and directly match a structure in memory. When these two structures are verified as identical, the sentence is comprehended.

Perceptual data compared to these high level model des-

fig. 10

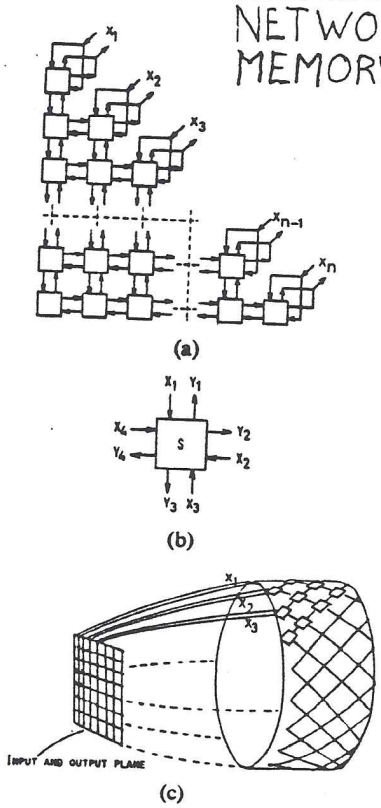


A SCHEMATIC DIAGRAM OF A POSSIBLE ORGANIZATION OF MEMORY.



Examples of entities. (a) Entity representing association of Fig. 8 and its image. This kind of entity is used in pattern recognition experiment. (b), (c) These entities are used in experiments concerning concept formation.

B. THE ASSOCIATRON; ASSOCIATIVE NETWORK WITH LOCALIZED MEMORY.



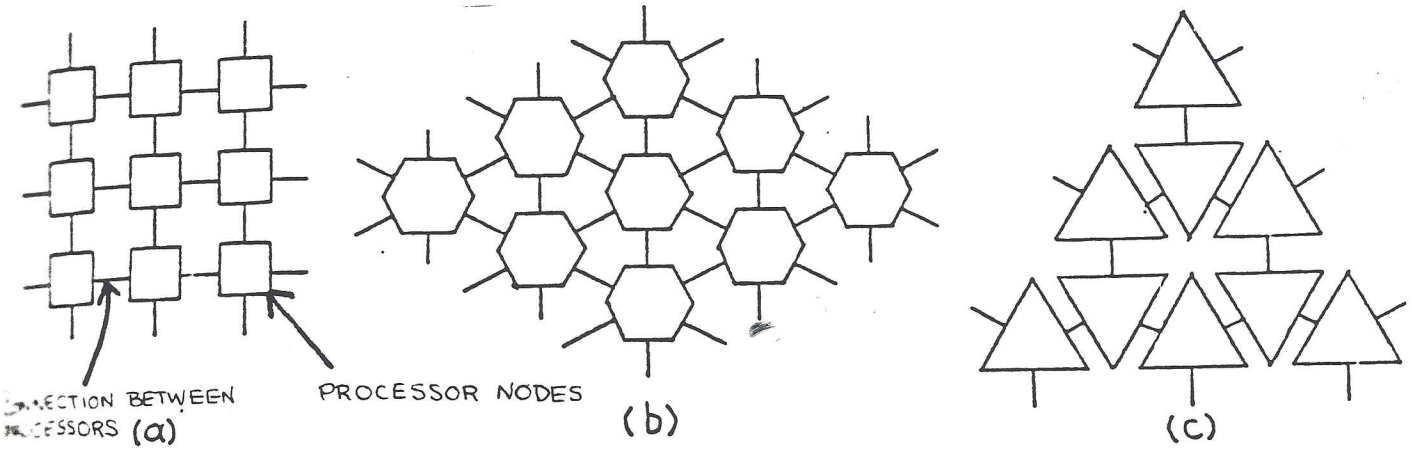
Hardware realization. (a) Associatron constructed as iterative circuit. (b) Element of iterative circuit is simple automaton whose function is described by (25). (c) Triangular structure of (a) changed to cylindrical structure.

B. THE ASSOCIATRON AS A REGULAR NETWORK. (NAKANO, K 1972)

criptions favor HAM over TLC. HAM's links have weights. The more related two things are, the higher the weight. In this way, a match can more easily be established for things that are more related. This ability to recognize more related things faster has been shown in many experiments to be true of human perception. For example it has been found that subjects can verify that a bear is an animal, faster than a bear is a mammal(Rips, C.J. Shoben, E.J. and Smith, E.E. 1973). In TLC, on the other hand, bear would be closer to mammal than animal and would therefore give a longer search time for animal than mammal (Klatzky, R.L. 1975). This analysis, that related things are inherently closer to more closely related things in HAM than TLC, depends on the high level descriptions of the models directly reflecting their implementation in the brain. The implementation notes for these models are absent. Although these semantic nets do not attempt to describe how they would be represented in the brain, and do not agree with all experimental data, they are clear, simple, and provocative demonstrations.

As a reaction to Lashley's experiments (Lashley, K.S. 1929), memory models in which specific memories are spatially distributed have been considered exciting. Lashley's experiments do not conclusively show that memory is distributed rather than redundant in the brain. Distributed memory

fig. 11

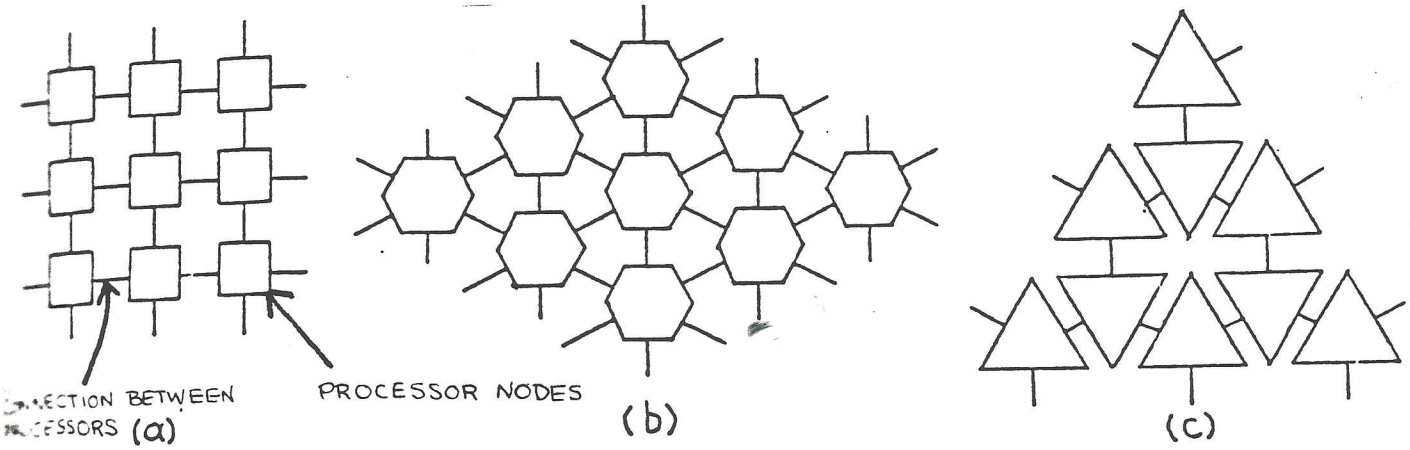


THREE TYPES OF REGULAR NETWORKS.
(a) RECTANGULAR, (b) HEXAGONAL, (c) TRIANGULAR NETWORKS.

KUNG'S COLLATED EXAMPLES OF ALGORITHMS WHICH CAN BE IMPLEMENTED EFFECTIVELY ON DISTRIBUTED NETWORKS. (KUNG, H.T., 1979)

ALGORITHM TYPES	EXAMPLES
SYSTOLIC ALGORITHMS USING:	
1-DIM LINEAR ARRAYS	REAL-TIME FIR-FILTERING, DISCRETE FOURIER TRANSFORM (DFT), CONVOLUTION, MATRIX-VECTOR MULTIPLICATION, RECURRENCE EVALUATION, SOLUTION OF TRIANGULAR LINEAR SYSTEMS, CARRY PIPELINING, SORTING, PRIORITY QUEUE, CARTESIAN PRODUCT, PIPELINE ARITHMETIC UNITS
2-DIM SQUARE ARRAYS	PATTERN MATCHING, GRAPH ALGORITHMS INVOLVING ADJACENCY MATRICES, DYNAMIC PROGRAMMING FOR OPTIMAL PARENTHESIZATION
2-DIM HEXAGONAL ARRAYS	MATRIX PROBLEMS (MATRIX MULTIPLICATION, LU-DECOMPOSITION BY GAUSSIAN ELIMINATION WITHOUT PIVOTING, QR-FACTORIZATION), TRANSITIVE CLOSURE, DFT, RELATIONAL DATABASE OPERATIONS
TREES	SEARCHING ALGORITHMS (QUERIES ON NEAREST NEIGHBOR, RANK, ETC., SYSTOLIC SEARCH TREE), PARALLEL FUNCTION EVALUATION, RECURRENCE EVALUATION
SHUFFLE-EXCHANGE	FAST FOURIER TRANSFORM, BITONIC SORT

fig. 11



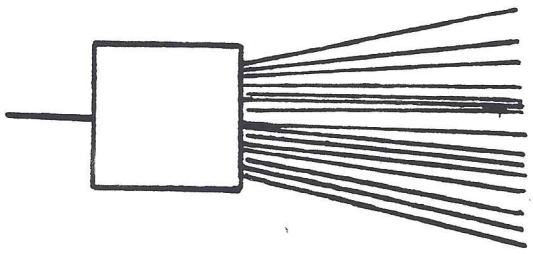
THREE TYPES OF REGULAR NETWORKS.
(a) RECTANGULAR, (b) HEXAGONAL, (c) TRIANGULAR NETWORKS.

KUNG'S COLLATED EXAMPLES OF ALGORITHMS WHICH CAN BE IMPLEMENTED EFFECTIVELY ON DISTRIBUTED NETWORKS. (KUNG, H.T., 1979)

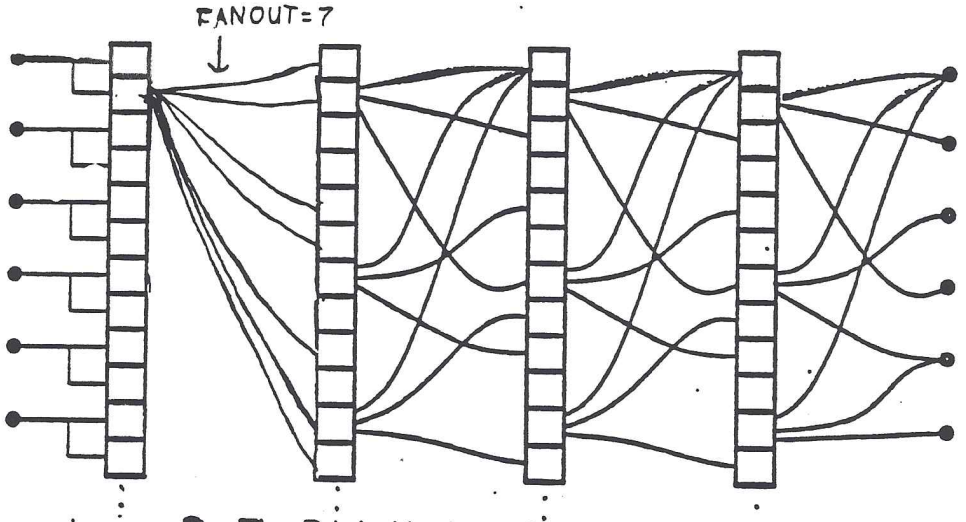
ALGORITHM TYPES	EXAMPLES
SYSTOLIC ALGORITHMS USING:	
1-DIM LINEAR ARRAYS	REAL-TIME FIR-FILTERING, DISCRETE FOURIER TRANSFORM (DFT), CONVOLUTION, MATRIX-VECTOR MULTIPLICATION, RECURRENCE EVALUATION, SOLUTION OF TRIANGULAR LINEAR SYSTEMS, CARRY PIPELINING, SORTING, PRIORITY QUEUE, CARTESIAN PRODUCT, PIPELINE ARITHMETIC UNITS
2-DIM SQUARE ARRAYS	PATTERN MATCHING, GRAPH ALGORITHMS INVOLVING ADJACENCY MATRICES, DYNAMIC PROGRAMMING FOR OPTIMAL PARENTHESIZATION
2-DIM HEXAGONAL ARRAYS	MATRIX PROBLEMS (MATRIX MULTIPLICATION, LU-DECOMPOSITION BY GAUSSIAN ELIMINATION WITHOUT PIVOTING, QR-FACTORIZATION), TRANSITIVE CLOSURE, DFT, RELATIONAL DATABASE OPERATIONS
TREES	SEARCHING ALGORITHMS (QUERIES ON NEAREST NEIGHBOR, RANK, ETC., SYSTOLIC SEARCH TREE), PARALLEL FUNCTION EVALUATION, RECURRENCE EVALUATION
SHUFFLE-EXCHANGE	FAST FOURIER TRANSFORM, BITONIC SORT

and redundant memory are both implied by his experiments. However, distributed memory has been the favored explanation of mass action experiments. Spatially distributed knowledge is usually explained by variations of Hebb's "field theory" on the neural level, and as associative memory on a higher level. Arguments for associative memories with calculations have made them attractive as a conceptual tool in brain modeling (Palm, G. 1980). If the memory encoding is somewhat more detailed or redundant than is needed for a task, and part of it is ablated, then the task may still be retrieved.

fig. 12 NETL (FAHLMAN, S.E., 1979)



A. FANOUT ELEMENT



B. The Basic Hashnet Arrangement
(MOST wires omitted for visibility.)

C.

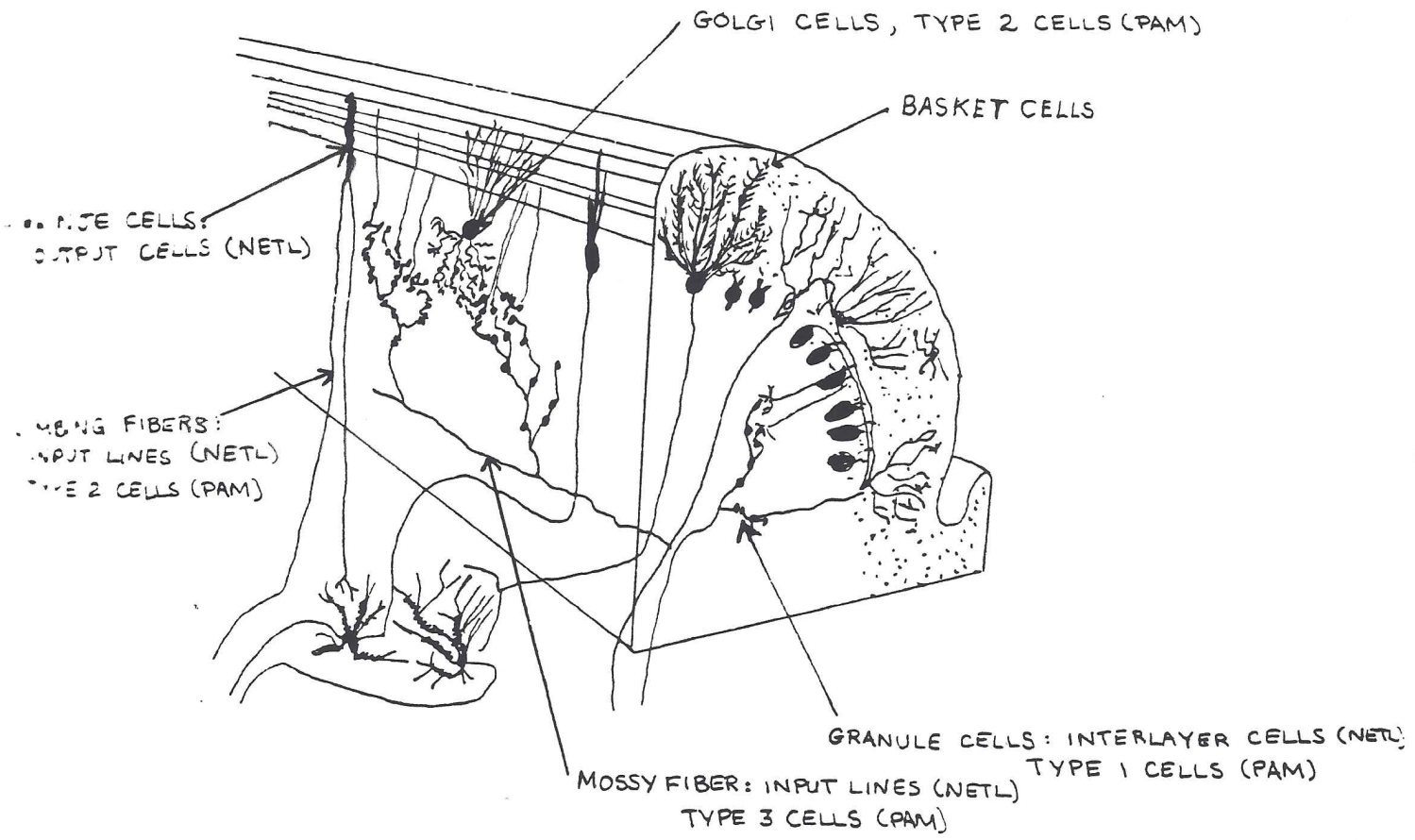
Fanout	15
Input Terminals	1000
Output Terminals	1000
Cells / Layer	2000
Fraction of Cells Used	.5

	In Empty Network: ONLY ONE INPUT CELL ACTIVE		In Full Network: MANY INPUT CELLS ACTIVE	
Layer	Cells Reached	Prob. of Blocking	Cells Reached	Prob. of Blocking
0	2	0.998000994	2	0.998000994
1	29	0.97044553	29	0.97044553
2	391	0.63977298	206	0.79985812
3	1893	2.46822253E-3	1076	0.0496812095
4	2000	4.20163906E-13	1964	6.44511867E-7
5	2000	9.3314245E-14	1998	3.05904567E-7
6	2000	9.3314245E-14	1998	3.05904567E-7
7	2000	9.3314245E-14	1998	3.05904567E-7
8	2000	9.3314245E-14	1998	3.05904567E-7

Two Hardware Implemented Models of Memory

Ambitious attempts have been made to integrate associative memories and semantic networks in hardware. Probably the earliest of these attempts, the Associatron (Nakano, K. 1972)(Fig. 10), is a simple regularly connected network. Such regular networks (Fig. 11) are becoming extremely popular for general research in distributed algorithm test beds (Kung, H. T. 1979). In this case, areas of the network are devoted to specific things. Bit patterns based on different stimuli for representing a thing are all convolved on an area. The idea is, that if the area has many, many more bits than the minimal representation for the thing, then it is probable that its different representations will not collide on the memory space. In this way, if any one of the patterns (or parts of several) are tested against memory, the memory can be tuned to elicit a match response. Part of the representation for a thing will be enough to retrieve the whole item.

fig. 13



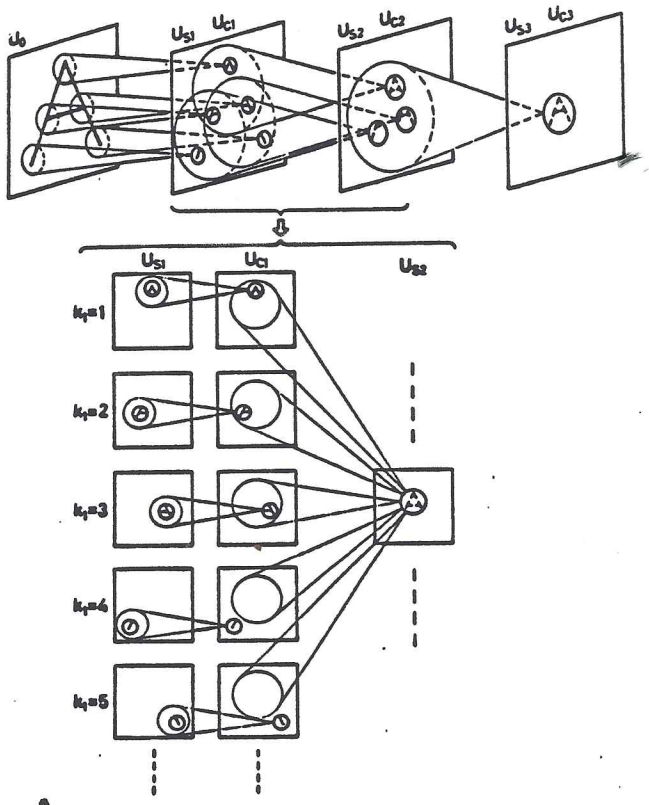
ADAPTED FROM C.A. FOX, 1962.
HYPOTHETICAL CORRESPONDANCES FOR CEREBELLAR CORTEX
AND TWO MEMORY MODELS. NETL (FAHLMAN, S.E., 1979)
SEE D. MARR, 1978 FOR DETAIL MODEL OF CEREBELLAR CORTEX..

Nakano's model is a brute force associative memory (Fig. 10). It does a nice job of showing how triplet or n-tuplet links between concepts can be used to elicit each other, how noise can be accomodated, and how redundant encoding can be integrated into an associative mapping. However, it still has localized memory. The implementation is only able to demonstrate trivial discrimination tasks.

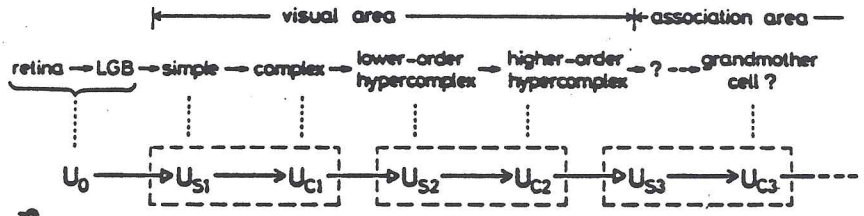
NETL (Fahlman, S.E. 1979, Fahlman, S.E. 1974) is an ambitious plan to create a hardware semantic network which is capable of storing enough facts to make it useful in several domains. One million elements is the number calculated to be necessary to achieve this goal. In this memory model, each element has the potential to connect to any other element in the network. A novel and attractive scheme is used to perform this interconnective function (Fig. 12) (Fahlman, S.E. 1980). A multilayered set of Fanout Elements is connected to the memory elements. Each Fanout Element input at the top of the interconnection network is attached to two memory elements. Each output layer element is connected to two memory elements. Internally, the 7 layered interconnection scheme uses random connections. An interconnection element has a fanout of 15. Each of these output wires from one layer are connected to random elements at the following layer. The probability for not being able to make a connection from any one memory element to any other element in

fig. 14

SCHEMATIC DIAGRAM ILLUSTRATING THE INTERCONNECTIONS BETWEEN LAYERS IN THE NEOCOGNITRON (FUKUSHIMA, K., 1980)



A. An example of the interconnections between cells and the response of the cells after completion of self-organization



B. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

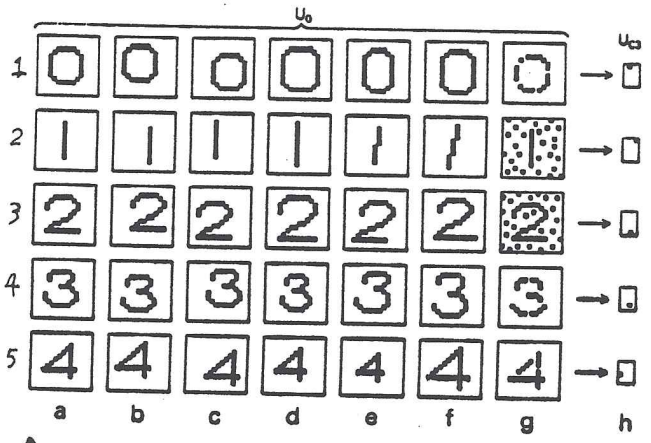
C. LAYERS ARE EXCITATORY AND MODIFIABLE.

S LAYERS ARE INHIBITORY

→ modifiable synapses

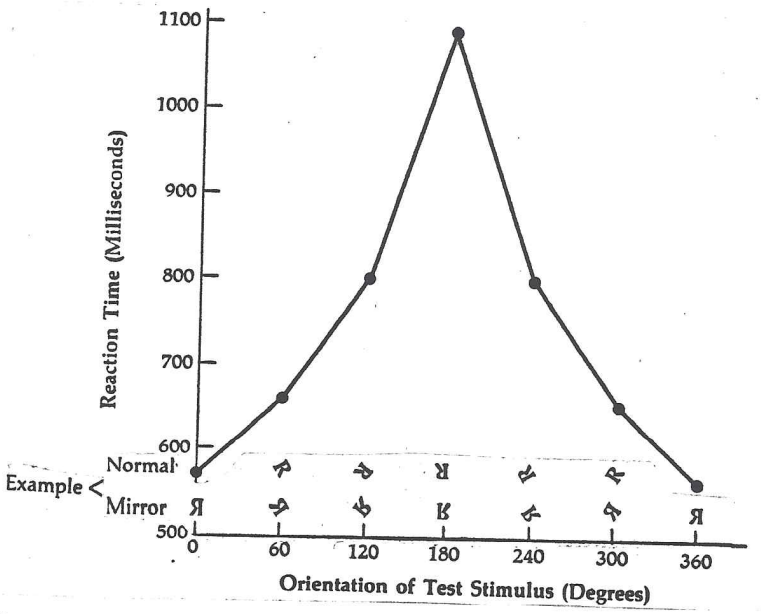
→ unmodifiable synapses

fig. 15



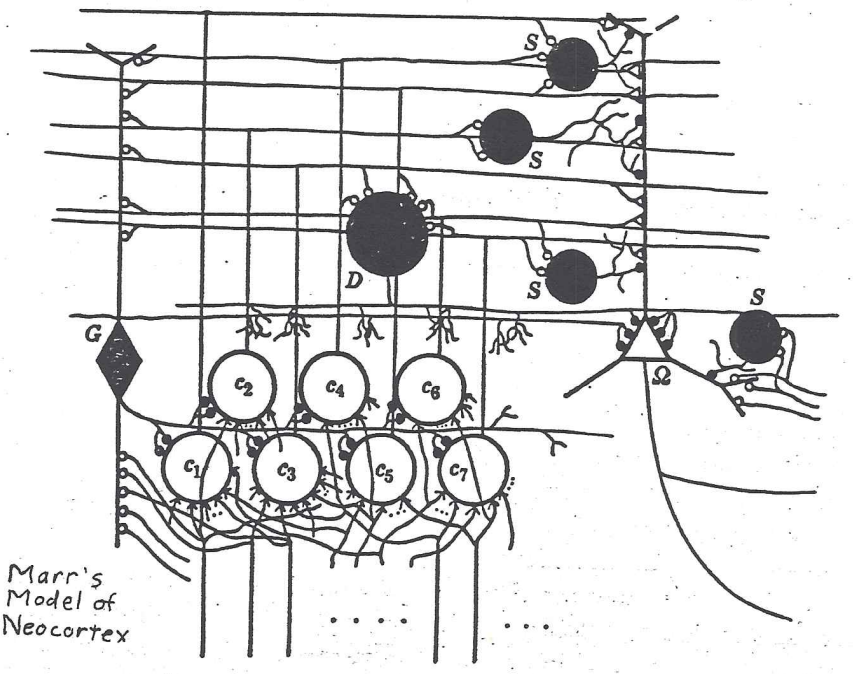
A. Some examples of distorted stimulus patterns which the neocognitron has correctly recognized, and the response of the final layer of the network

NOTE: IN TABLE , 2e AND 2f ARE "ROTATION RECOGNITION" IN NEOCOGNITRON (FUKUSHIMA, 1980)



B. Reaction time for deciding whether a letter is presented in its normal form or as a mirror image of its normal form, as a function of the degrees of rotation of the letter in the plane of presentation. Below the graph are examples of letters at each rotation value [After Cooper and Shepard, 1973.]

fig. 16



basic neural model for diagnosis and interpretation. The evidence cells $c_1, \ldots, c_7$ are codon cells with Brindley afferent synapses. The G-cell controls the codon cell threshold: it uses negative feedback through its ascending dendrite to keep the number of codon cells active roughly constant. Its descending dendrite samples the input fibres directly, thus providing a fast pathway through which an initial estimate is made. The other cells and synapses are as in figures 3 and 5(2). (MARR, D., 1970)

this scheme, even if most of the memory elements are using the interconnection network, is .04.

NETL's semantic network, like HAM, uses many link types to encode relationships between nodes. This semantic network suffers from the same limitations as HAM. Its interconnection network, on the other hand, can incur bullet wounds without degrading significantly in effectiveness. NETL's interconnection geometry is appealing for the simplicity of its assumptions, its power, and its correlation with the brain.

Although Fahlman does not attempt it, the interconnect topology can easily be mapped onto the anatomy of the Cerebellar Cortex (Eccles, J.C. 1967) (Fig. 13). In such a scheme, the Mossy fibers and Climbing fibers are the input lines, the Granule Cells are the inter-layer cells; their parallel fibers are their outputs and the Purkinje Cells the output cells of the interconnection scheme.

Layered Models of Memory

The brain is organized in stereotyped structures with a common theme of interconnected layers (Eccles, J.C. 1973, Carpenter, M. B. 1976). Several people have put forward models based on this theme. The layered nature of the visual system has motivated models of how organisms perform simple internal perceptual tasks (Selker, E.J. 1979, Moreno-Diaz, E.R. 1980 et al).

The NEOCOGNITRON is a layered model of spatial memory in which alternating inhibitory and excitatory layers work to recognize patterns (Fig. 14) (Fukushima, K. 1980). The system claims to be able to recognize patterns independent of position and rotations. This is not strictly true. The system can recognize a slightly tilted or shifted pattern. The claim of perception being unaffected by rotation is not strongly supported by data for organisms in any case. Mental rotation experiments with people do not indicate that the recognition of a rotated character would be the same as an unrotated character. It has been shown that the time it takes to recognize letters is proportional to the degree of the letter's rotation (plus the recognition time for an unrotated letter) (Cooper , L.A., and Shepard, R.N. 1973)(Fig. 15).

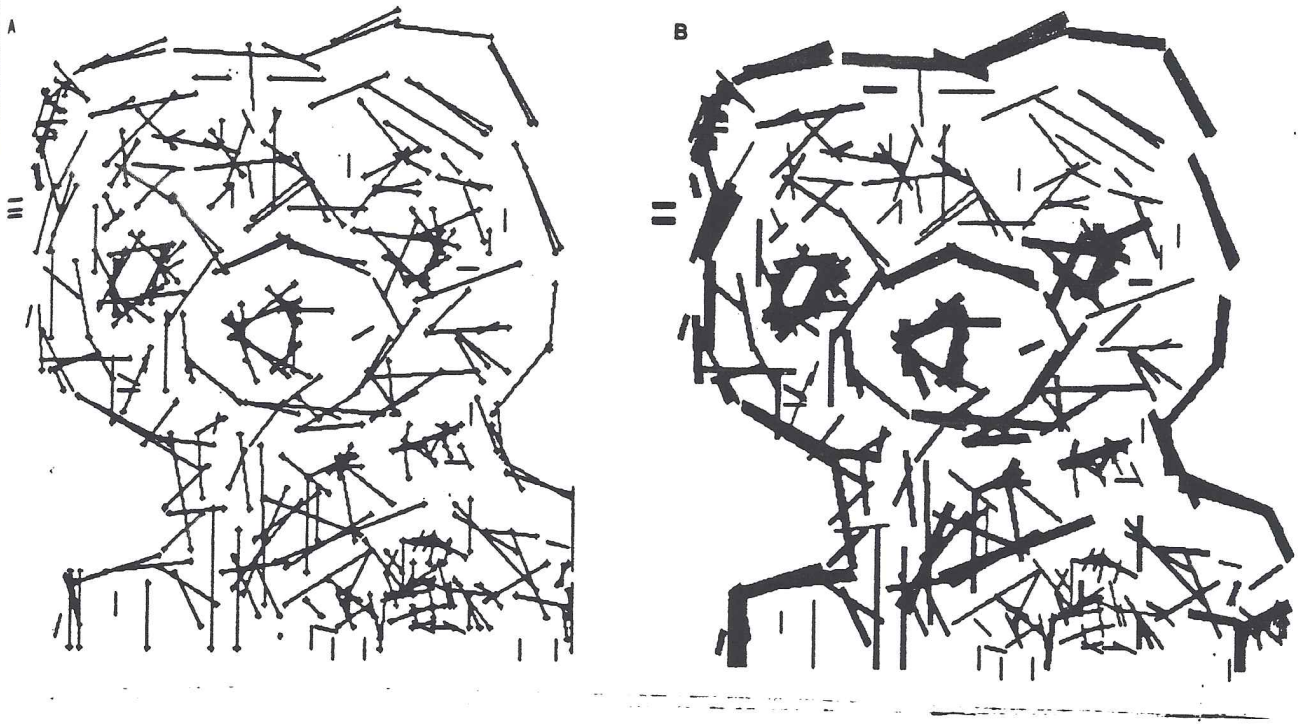
The structure of the Cerebellar Cortex as described by Eccles (Eccles, J.C. 1967) has inspired several connectionistic memory models. Many such models depend on stereotyped circuits much like Hebb's. One layered model which has attracted a lot of attention is David Marr's theory of the cerebral neocortex (Marr, D. 1970)(Fig. 16)

This is a serious attempt to put a memory model into the context of the known modern neural structure of the brain. The concept that redundant information from the sensory systems decreases the work a memory model has to perform is approached. Marr describes early sensory processing as eliminating excessive redundant information in sensory data. He describes the outputs of the system as being made up of subevents called "features". A large body of literature based on neurophysiological data supports this claim (Hubel, D.H., and Wiesel, T.N. 1962 et al).

Marr's model presents an implementation for a cluster theory memory model. Its implementation is described in terms of known anatomical and physiological data about the brain. The model uses the concept of memory evidence which allows recall of a memory based on a threshold. In this way, this model responds much like the Associatron (Nakano, K. 1972)(Fig. 10). One problem with the theory is that it does not include data which necessitates non-localized memo-

ry for reliability purposes. Data such as a man having a measurable IQ of 160 after his prefrontal lobe was removed are hard to explain with such a connectionistic model (Hebb, D.O. 1939).

A recent disertation (Maclaren, L.S. 1979) describes a layered, neural model which avoids this problem. Maclaren's work uses Marr's goals and tries to use a randomly connected layer of neurons with surround inhibition to implement a clustering memory. The choice of cluster, or features is critical. Like the NEOCOGNITRON (Fukushima, K. 1980), Maclaren's model uses a refining set of layers and relies on layers being different from each other. Neither of these people attempt to motivate this requirement from data about neuroanatomy.



The so-called primal sketch is shown in three of its aspects; each is a representation of intensity changes in a gray-level image such as appears on page 3. Its creation constitutes the earliest stage in the authors' theory of visual information processing. Sketch A shows only "edge-assertions": each line represents the position and orientation at which a change in intensity is found. The cross-bars at the ends of each line show the terminations of the change. This is not to say that each line necessarily denotes a sharp edge in the gray-level array, but rather that each denotes the existence of a gradient in intensity. Sketch *B* adds information about differing contrasts: each line in *A* is transformed into a set of parallel lines in proportion to the logarithm of the intensity change. In other words, a large magnitude of transition from light to dark or dark to light leads to a bolder edge-assertion. Notice, accordingly, that the margin of the bear tends to have more prominent assertions than those to be found within. In the displays shown here, the spacing between parallel lines in a set is simply proportional to their length. For short assertions, therefore, the multiple lines have tended to overlap, so as to create the impression of a single, thick bar. Sketch *C* shows the fuzziness of each of the edge assertions — that is to say, the widths of the variations in intensity, as opposed to their magnitudes. The thicker sets of lines now tend to lie in the interior of the image, which reflects the circumstance that broad gradients in shading tend not to appear at its edges. As held in computer storage, the primal sketch includes the information shown in *A*, *B*, and *C* alike: that is to say, the positions, directions, magnitudes, and spatial extents of intensity gradients in the gray-level array.

fig. 17

TWO AND ONE HALF DIMENSION SKETCH (MARR, D. AND NISHISHIHARA, H.K., 1978)

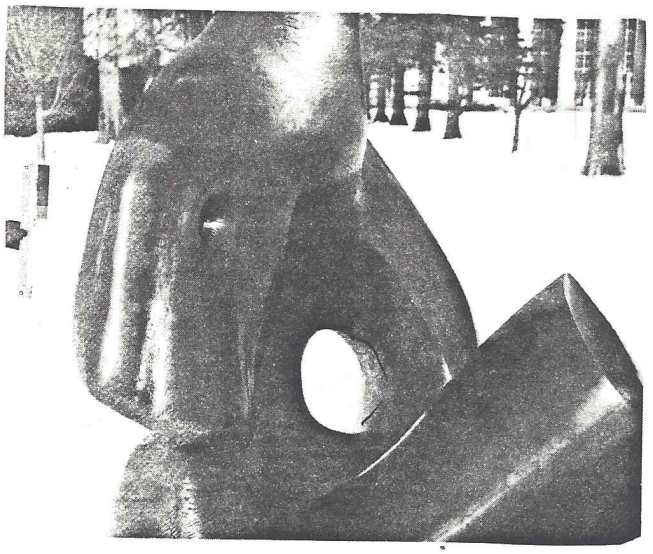
Memory Features Encode Knowledge

The consensus of neurophysiological and psychological data describes memory as well as sensory data as being represented in internal representations having some predictable attributes. Some of these attributes give possible representational models stereotyped characteristics. Experimental data points to at least two levels of internal representations: physiological and perceptual, which we use for spatial data encoding.

All organism data representations seem to be "feature" encoded. Frog retinas have explicit sensors which are activated only by specific fly type movements (Lettvin, J.Y., Maturana, H., McCulloch, W.S. and Pitts, W.H. 1929). Their retinas are made up of several feature-specific receptor types. Intracellular recording in the Lateral Geniculate Nucleus (LGN) and Visual cortex of cats (Hubel, D.H. and Weisel, T.N. 1962) has allowed us to learn that visual stimuli are feature encoded in the visual pathway. Cells in the LGN are receptive to specific shapes and orientations of shapes with on or off, center positive or negative contrasts. This feature encoding is in some sense an elimination of redundant data in the image imposed on the retina. The cells elicit maximum response for complicated visual scenes and change of stimuli. Blank areas of visual stimuli

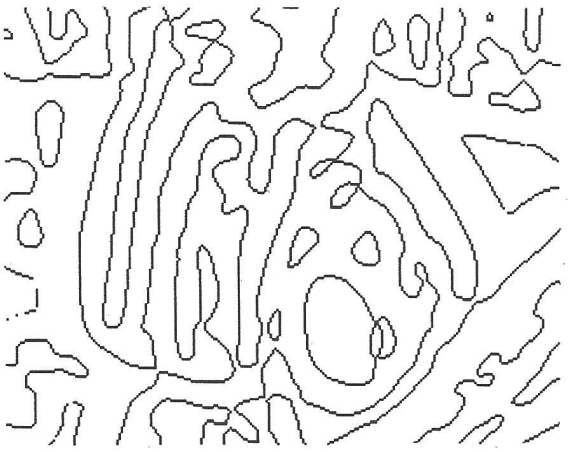
fig. 18

a.



b.

c.



d.

6. The image (a) has been convolved with $\nabla^2 G$ having $w = 2\sigma = 6$, 12 and 24 pixels. These filters span approximately the range of filters that operate in the human fovea. (b), (c), and (d) show the zero-crossings thus obtained. Notice the fine detail picked up by the smallest. This set of figures neatly poses our next problem--how does one combine all this information into a single description?

elicit little or no cell response.

Experiments have also discovered data reducing feature encoding at the perceptual level. Hearing studies (Eimas, P.D., and Corbit, J.D. 1973) found that people (even 2 to 3 month old infants) have a 30 millisecond voiced onset time detector. This detector allows us to distinguish between the voiced phoneme "ba" and the unvoiced phoneme "pa" with a predictable threshold. This surprising data indicates that people have a 30 millisecond voiced onset detector, which they use to classify "ba-pa" sounds. A sound with a voice onset time of less than 30 milliseconds is consistantly heard as ba! Likewise such sounds with a longer voice onset time are interpreted as "pa". In these ways sensory information is seen to be quickly converted and manipulated in terms of feature based representations in organisms.

A body of experiments aimed at learning about how people encode spatial knowledge has revolved around city street maps and directions (Kuipers, B. 1978). Such experiments have found that people encode spatial knowledge in terms of features such as "landmarks", "nodes", "paths", "edges" and "districts" (Lynch, 1960). People make mistakes about distances and distort actual positions, but do not distort the structure of the two-dimensional relationships between landmarks.

Based on his experiments with cognitive map formation, Kuipers describes (Kuipers, B. 1978) a model for spatial knowledge representation based on networks containing routes, topological structure, relative position, and region segmentation. The TOUR model is a program in LISP which has a current position pointer and uses built-in inference rules to manipulate knowledge represented in its internal language (Kuipers, B. 1978).

Whether a recreation of the cartesian coordinate relationships between spatial entities, or explicit feature encoding of data, knowledge models have internal representations. Many people have advanced the idea of human memory being feature based on a plausibility argument (Katz, J.J., and Fodor, J.A. 1963, Miller, G.A. 1972, Osgood, C.E. 1952 et al.) The Pandemonium model (Selfridge, O.G. 1959) is an early demonstration of a model idea which encoded knowledge into features. This pandemonium model has a basis set of "demons" into which it encodes stimuli. It has another set of cognitive "demons" which it classifies the encoded stimuli into.

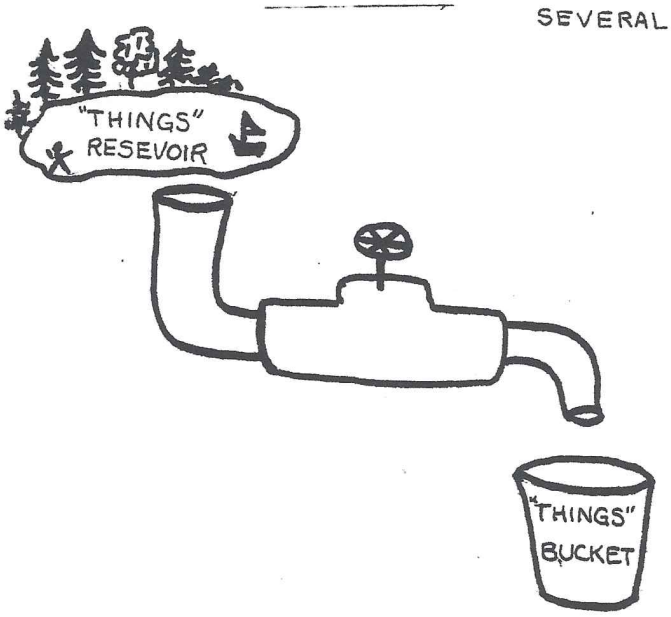
Utilizing observations of the ever constant redundancy in data, compact internal representations can be designed which are sensitive to critical features and are somewhat insensitive to noisy data (Marr, D. 1970). An internal

model can also make it easier for a model to search its solution space (Nilsson, N. J. 1971).

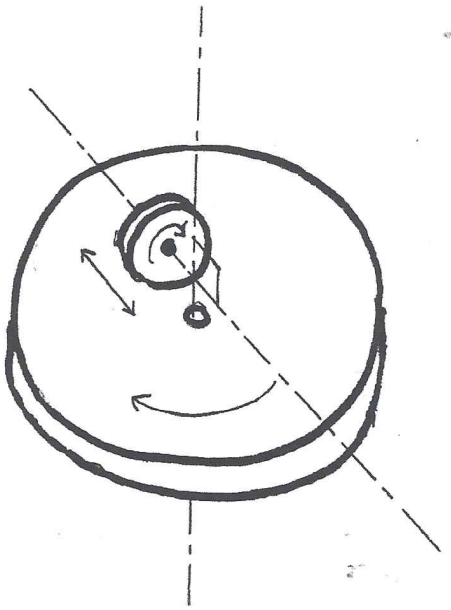
A spatial data representation dubbed the "two and one half dimension sketch" (Marr, D. 1978) is such a model (Fig. 17). In his representation, Marr presents images as contrast lines which encode width of light level gradients and magnitude of light level gradients. The representation has a pleasing resemblance to data representation which intracellular recording in cats suggest. It is also an excellent example of reduction of redundancy in data to significant, noise tolerant features. These edge features hold all picture data. In a related article, it is shown that by using 5 edge filter feature detectors, all spatial information can be encoded (Marr, D., Hilderith, E. 1979) (Fig. 18). This description of a stroke based spatial encoding is a recurring theme. The idea of a system which discovers its basis line or stroke alphabet is much more powerful than a system which has a preset alphabet for spatial encoding. Another such system for edge encoded spatial representations is described in a recent model (Chian, C. 1980).

The above models are described in terms of high level semantics. These systems are like production systems in which actions are prescribed for given inputs. Bringing concepts of internal feature encoded memory representation

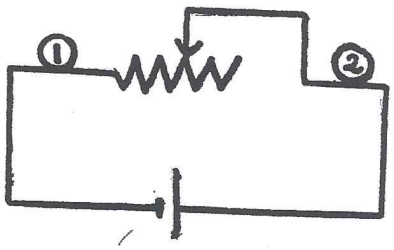
fig. 19 VARIABLE STRENGTH DESCRIBED IN SEVERAL TECHNOLOGIES



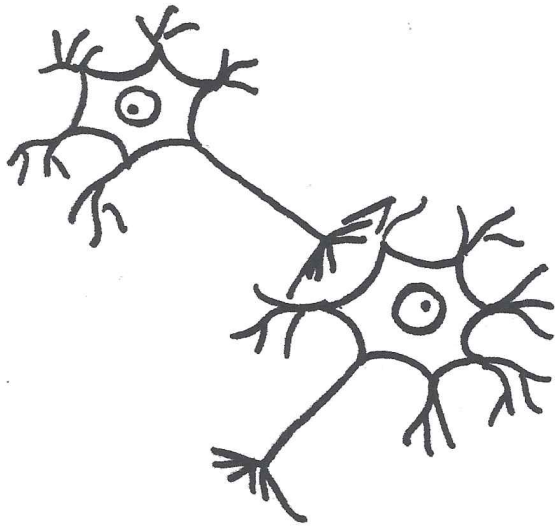
STRENGTH OF ASSOCIATION IS CONTROLLED BY VALVE..



STRENGTH OF ASSOCIATION OF TWO PERPENDICULAR WHEELS IS RELATED TO WHERE OR IF THEY ROLL ON EACH OTHER.



STRENGTH OF ASSOCIATION OF ELECTRONS AT TWO ENDS OF WIRE IS RELATED TO THE RESISTANCE THEY HAVE TO CROSS.



STRENGTH OF ASSOCIATION RELATED TO CELL OUTPUT, CELL SYNAPSES, INHIBITORY INFLEUNCE AND NEUROTRANSMITTERS AVAILABLE.

down to neural implementation is ultimately useful. One recent description of pattern discrimination (Foster, D. H. 1980) describes spatial features and spatial representations in terms of probability distributions. The language in which we describe models may be exceedingly important as we search for insights into the workings of the brain and how to model it.

Maclaren's dissertation (Maclaren, L.S. 1979) attempts a serious approach to brain theory using production system terminology. From a simple layered neural model mentioned briefly above, he develops a production system based model. Such attempts to bridge the terminology and technologies of Artificial Intelligence, and Brain theory and Cognitive Psychology are the fulcrums for comparing ideas from these different disciplines.

It is an exciting prospect to consider constructing a system which will have the ability to use the knowledge it acquires in a flexible manner; forming new schemata based on cluster analysis to capture underlying relationships.

Such systems with bootstrapping have received a lot of interest. In describing such systems with so-called unsupervised learning, Soloway states that he believes uniform and flexible data representation is critical to such an ap-

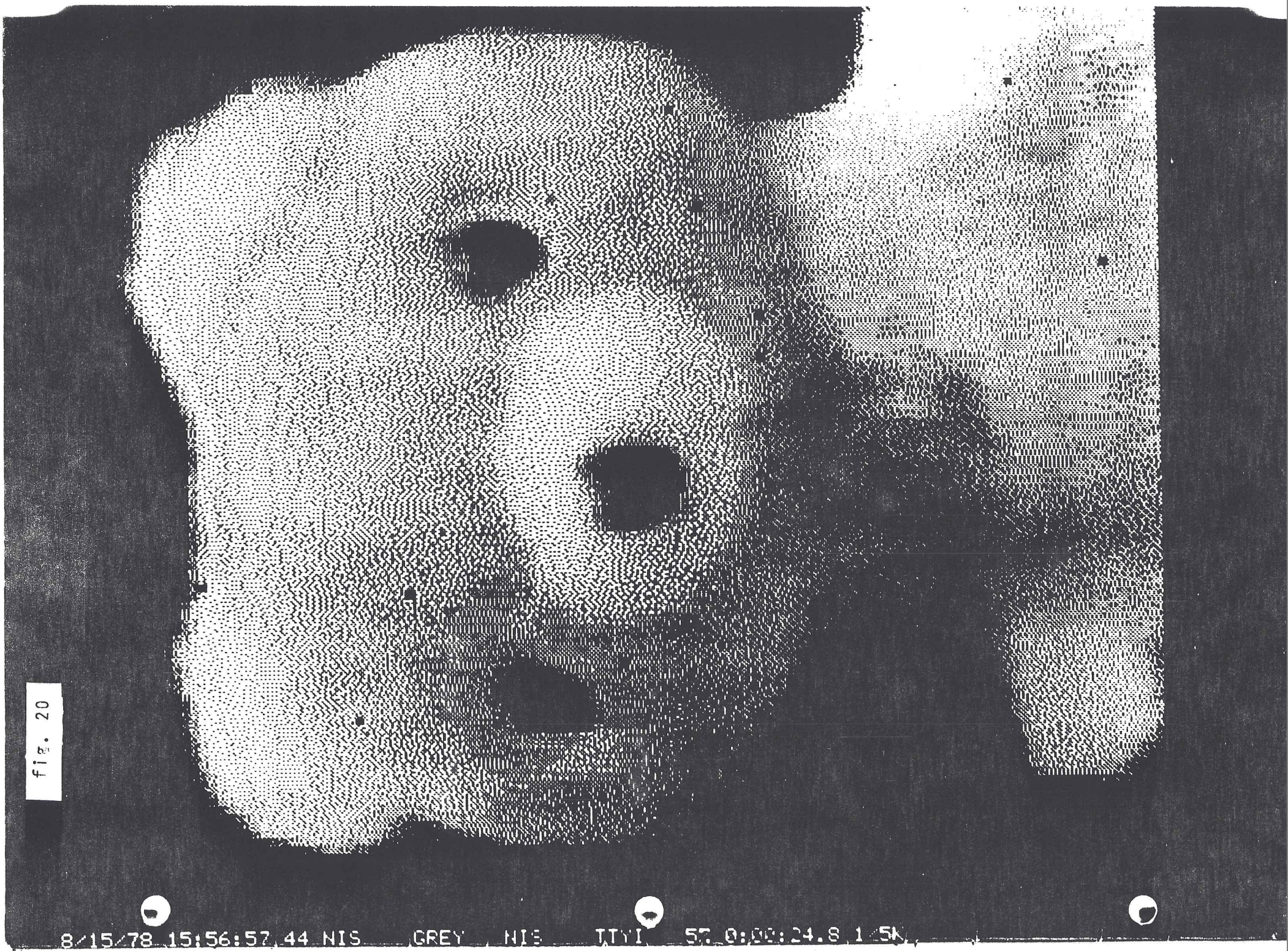
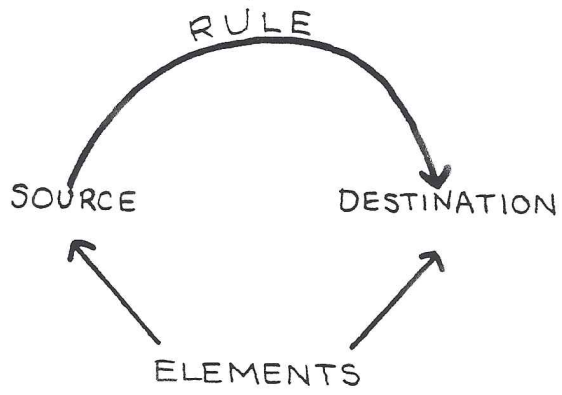
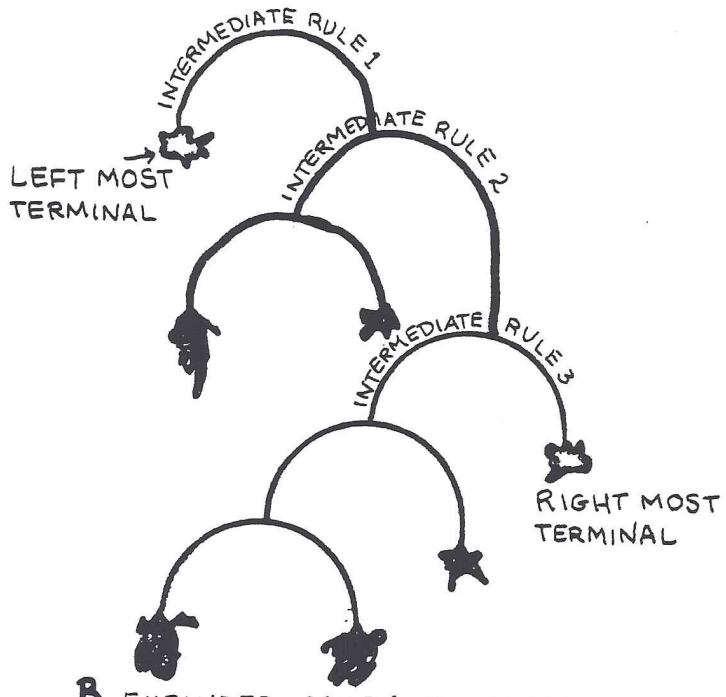


fig. 20

fig. 21



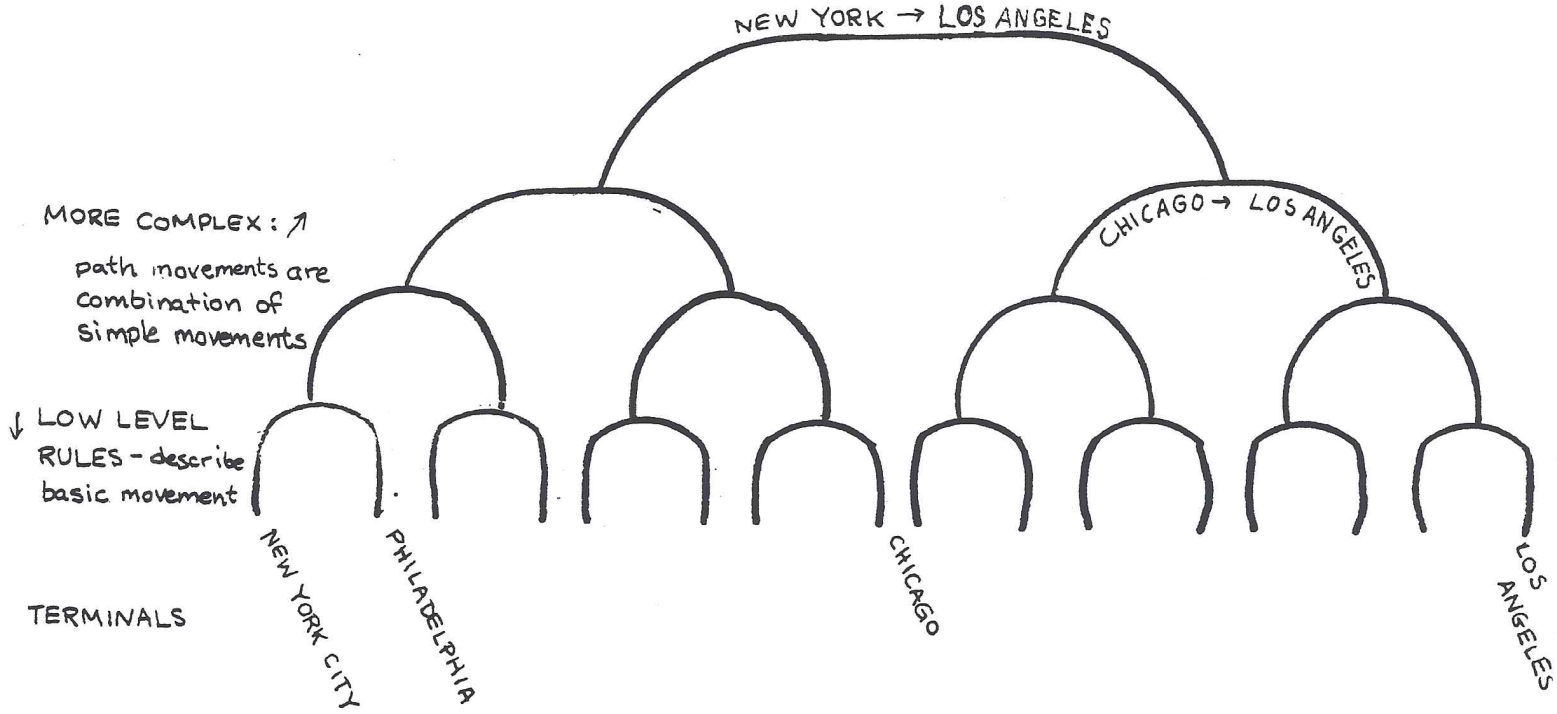
A. SCHEMATIC DIAGRAM OF PAM MEMORY PAIR



B. EXPANDED SCHEMATIC DIAGRAM OF A SEARCH IN PAM.

C. HIERARCHIES IN P.A.M.

THE SIMPLE NATURE OF DATA IN THIS EXAMPLE AND THE DETERMINISTIC CONNECTIVITIES ARE NOT CONSTRAINTS OF THE MODEL, BUT ARE SIMPLY USED FOR ILLUSTRATIVE PURPOSES.



proach (Soloway, E.M., Riseman, E.M. 1977).

Now that we have reviewed the relevant memory modeling literature, the remainder of this paper develops an organism memory model: + Paired Associative Memory (PAM). For a model to attempt to describe organism memory, it should be translatable into a neural implementation, and should take into account data from the scientific investigations in Psychology, Biology and Artificial Intelligence. It should then be able to model developed organism behavior as AI does using goals, hierarchies, clusters and Piagetian learning theory. As well, it should be an ablation tolerant neurally implementable theory. The model must allow similarly related memories to elicit each other, and be able to retrieve memory in features.

Its internal model should be designed to speak to issues of learning and recall as well being plausibly translatable into the brain's anatomy. As well, the memory model must be able to build upon itself to encode new things. Some believe that a uniform internal representation gives maximum flexibility. The language in which the model is formulated should allow it to be compared with other models. The following is the development of a simple model which grew from the background described above and two constraining assumptions.

PAM (Paired Associative Memory): Design Constraints

The following restrictions are imposed on our modeling effort simplified and directed it. Even the simplest living things make binary associations such as associating light with the direction to go. If all that a basic unit needs to do is associate two things with a variable strength, it can be modeled with almost any simple system. Such a unit is the building block of PAM. Fig. 19 proposes wheels, pipes and water, wires and stylized neurons as such units . However, we do not advocate the process of simply connecting up modeled neurons as though they were elements in electrical circuits (Daniels, J.D. 1978 et al). Connectionistic approches like these can easily lead to models with localized electronic circuit-like characteristics. Exclusive of some new ultra-radical experiments with self fixing Large Scale Integration (LSI) circuits (Brinton, J.B. 1979), people design electronic circuits to achieve the most signal processing with the fewest connections. In circuits designed with this philosophy, each connection is critical. Although complex things can be constructed in this way, it does not produce distributed or redundant self fixing circuits. To model organism memory, we believe that it is important to construct parallel probabilistically connected circuit.

Imagine that a stimulus pattern such as the visual sensory impression caused by Marr's teddy bear (Fig. 20) can elicit an internal memory of some sort. To be useful, such a representation, compact as it is, requires a context defined by other visual and nonvisual information. We imagine this information and virtual stimuli as being represented as patterns of cell activity on a layered network. These patterns we hypothesize, correspond to the connectivities of cells and other brain structure inputs. These information patterns associate with each other through cells which connect areas of patterns. Such cells develop through Hebbian growth. Say associations like this can have strength weightings and have as one of the associating elements another association. With these restrictions; that of only allowing binary, variable weight pattern associations and the requirement of distribution of information, a network which can simulate LTM can be constructed in which a hierarchy of associations can be built (Fig. 21).

We hold that even simple unlabeled associations can encode any information describable in a semantic model like HAM, TLC, or NETL. The relationships between associated "features", defined by labeled edges in other semantic networks, are explicitly described with weight thresholds and other associations between features in the PAM model. All knowledge in the system (including how features are related)

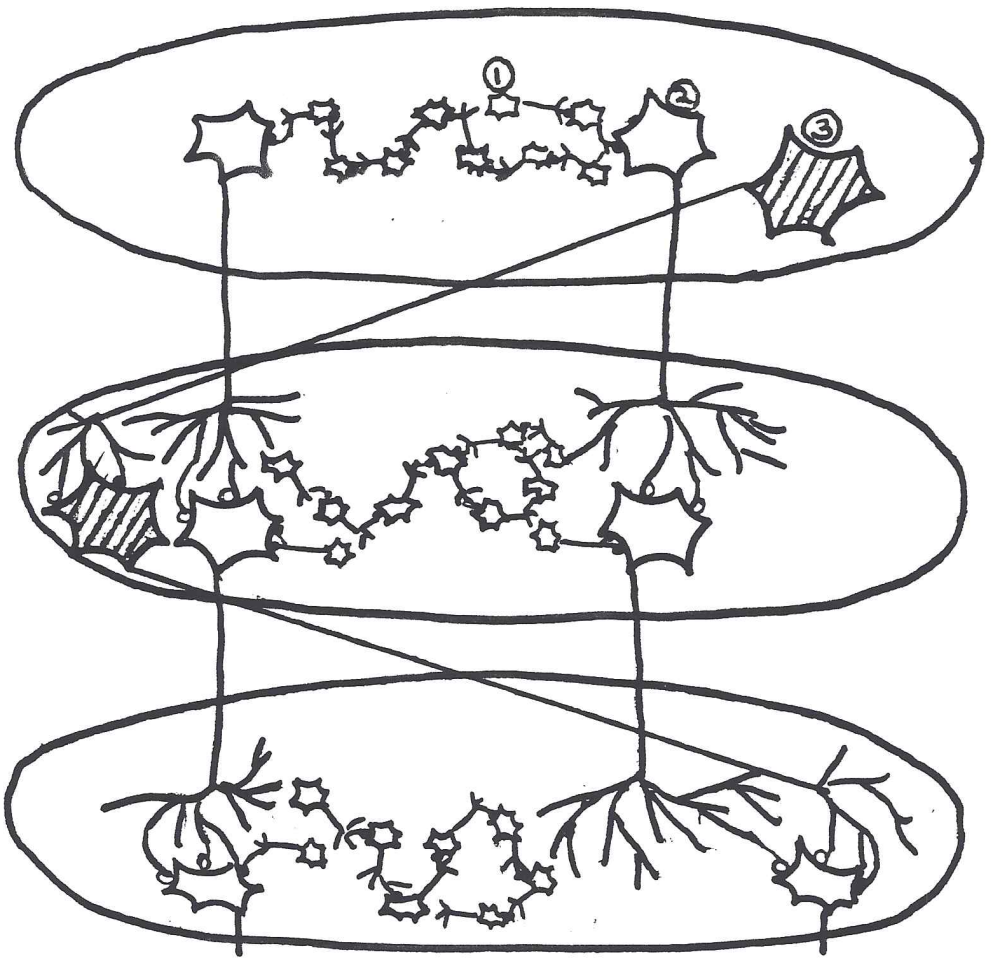


fig. 22

THE ABOVE DIAGRAM SHOWS THE THREE CELL TYPES IN THE NEURAL PAM:

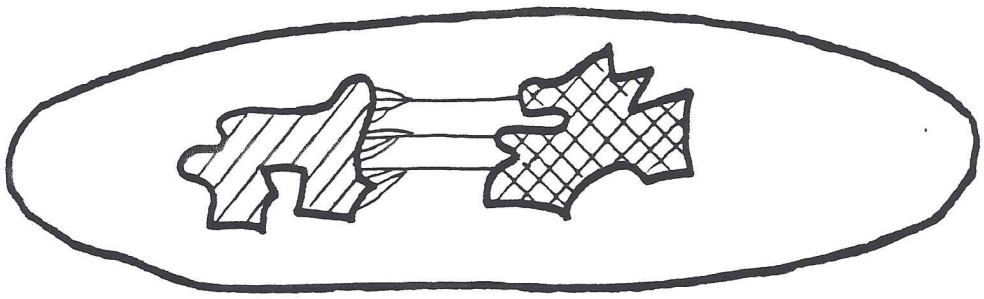
① CELL TYPE 1 - SMALL CELLS ON LAYERS INHIBIT SURROUNDING CELLS.

② CELL TYPE 2 - CELLS BETWEEN LAYERS GIVE SPREADING EXCITATORY OUTPUT.

③ CELL TYPE 3 - CELLS HAVE BOTH EXCITATORY AND INHIBITORY MODIFIABLE SYNAPSES. THESE PARALLEL FIBERS GIVE LONG RANGE INFLUENCES.

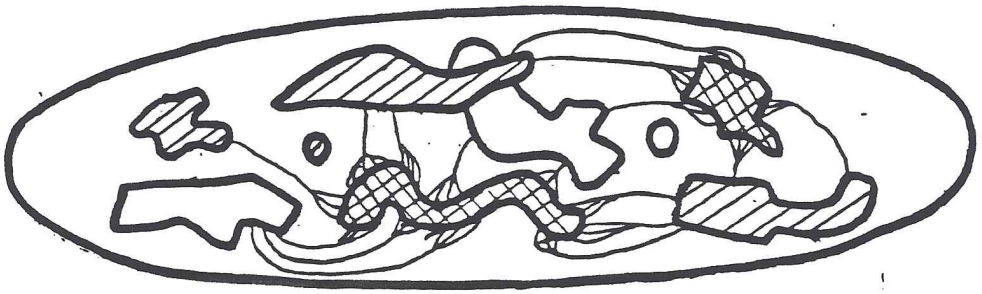
is proximity and weight encoded feature associations. Some features are described by stimulus patterns such as the visual impression of a teddy bear, others by some internal description of an action. Associations can also be features. This internal net representation will be described using a graph with weighted edges as a bootstrapping production system. Before we define this related, high level graph model more precisely, let us discuss a distributed architecture which we define as the neural network.

fig. 23

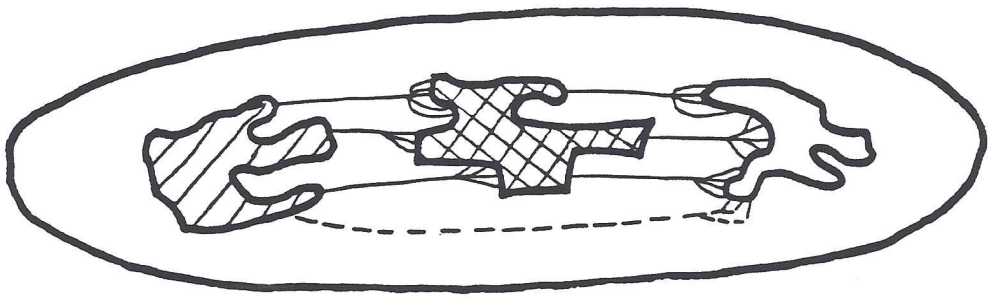


A.

SCHEMATIC DIAGRAM DENOTES LINED EXCITATION AREA BEING ACTIVATED BY CROSS-HATCHED EXCITATION AREA IN NEURAL NETWORK.



B.



C.

SCHEMATIC DIAGRAM OF B

IN DIAGRAM B, THINK OF ALL THE CROSS-HATCHED AREAS AS THE PLACES IN THE NETWORK WHICH HAVE HIGH EXCITATION FOR SOME PATTERN. IF EACH OF THE THREE TYPE AREAS DENOTE PATTERNS OF EXCITATION FOR DIFFERENT STIMULI; THESE ARE THE SYMBOLIC ENCODING PATTERNS FOR THEM. FOR ONE OF THESE MEMORIES WHEN EXCITED TO ELICIT ANOTHER, THEY MUST HAVE SUFFICIENT TYPE 3 CELLS CONNECTING THEM. IN THIS DIAGRAM: CLEAR AREAS RARELY CONNECT TO SLASHED LINE AREAS; LINED AREAS ALWAYS CONNECT TO CROSS-HATCHED AREAS; CROSS-HATCHED AREAS ALWAYS CONNECT TO CLEAR AREAS. IF LINED AND CLEAR AREAS GET EXCITED SIMULTANEOUSLY, THEY WILL FIND THEMSELVES ASSOCIATED THROUGH CROSS-HATCHED AREA; HEBBIAN GROWTH WILL TEND TO ASSOCIATE THEM BY FORMING NEW TYPE 3 CELL CONNECTIONS.

fig. 24

Feature / Source / destination / Value

```
(make-cando)
((twth (tw th) 1) (detour-brestof-a (detour-b restof-a) 2)
                  (startastbx (startof-a start-box) 6)
                  (startc (start-box route-c) 1)
                  (route-cgoal (route-c goal-box) 1)
                  (goalrest (goal-box restof-a) 4)
                  (reststart (restof-a startof-a) 4)
                  (restdet (restof-a detour-b) 1)
                  (startof-adetour-b (startof-a detour-b) 1)
                  (route-cstart-box (route-c start-box) 1)
                  (goal-boxroute-c (goal-box route-c) 1)
                  (restof-agoal-box (restof-a goal-box) 4)
                  (startof-arestof-a (startof-a restof-a) 4)
                  (start-boxstartof-a (start-box startof-a) 4)
                  (thsix (th six) 1)
                  (sixn (six n) 1)
                  (ntw (n tw) 1)
                  (ntw (n tw) 1)
                  (sixn (six n) 1)
                  (thsix (th six) 1)
                  (start-boxstartof-a (start-box startof-a) 4)
                  (startof-arestof-a (startof-a restof-a) 4)
                  (restof-agoal-box (restof-a goal-box) 4)
                  (goal-boxroute-c (goal-box route-c) 1)
                  (twth (tw th) 1)
                  (twth (tw th) 1)
                  (goal-boxroute-c (goal-box route-c) 1)
                  (restof-agoal-box (restof-a goal-box) 1)
                  (startof-arestof-a (startof-a restof-a) 4)
                  (start-boxstartof-a (start-box startof-a) 4)
                  (thsix (th six) 1)
                  (sixn (six n) 1)
                  (ntw (n tw) 1)
                  (ntw (n tw) 1)
                  (sixn (six n) 1)
                  (thsix (th six) 1)
                  (start-boxstartof-a (start-box startof-a) 1)
                  (startof-arestof-a (startof-a restof-a) 1)
                  (restof-agoal-box (restof-a goal-box) 1)
                  (goal-boxroute-c (goal-box route-c) 1)
                  (route-cstart-box (route-c start-box) 1)
                  (startof-adetour-b (startof-a detour-b) 1)
                  (twth (tw th) 1)
                  (twth (tw th) 1)
                  (startastbx (startof-a start-box) 6)
                  (detour-brestof-a (detour-b restof-a) 2)
                  (twth (tw th) 1)
                  (startastbx (startof-a start-box) 6)
                  (detour-brestof-a (detour-b restof-a) 2)
                  (twth (tw th) 1)
                  (startastbx (startof-a start-box) 6)
                  (detour-brestof-a (detour-b restof-a) 2)
                  (twth (tw th) 1))
"if you have any additions to %cando%; put in form:"
"(label (state1 state2) z) where z is importance"
"type nil when done with entries"
? nil
nil
```

fig. 25

A

```
(sethere)
start-box
"is presently where you are"

((th (497 470) 1) (six (470 440) 1)
                  (n (440 470) 1)
                  (tw (470 500) 1)
                  (start-box (186 230) 7)
                  (startof-a (158 180) 6)
                  (restof-a (112 192) 6)
                  (goal-box (101 241) 8)
                  (route-c (115 280) 4)
                  (detour-b (182 163) 4))
"input where you want to be"
? goal-box
goal-box

?
```

Feature
x coordinate
y coordinate
value

B

```
? (make-world)
((th (497 470) 1) (six (470 440) 1)
                  (n (440 470) 1)
                  (tw (470 500) 1)
                  (start-box (186 230) 7)
                  (startof-a (158 180) 6)
                  (restof-a (112 192) 6)
                  (goal-box (101 241) 8)
                  (route-c (115 280) 4)
                  (detour-b (182 163) 4))
"if you want rules made enter r"
? r
"if you have any additions, type them in with format"
"(things (x y)z) where z is current # connections"
"otherwise type nil"
? nil
nil
```

← explicit placement of features (see also crsr-world) and Travel

← allows implicit travels from *last setting new rule to *last

rules. This weight becomes larger as rules show themselves useful by being reused, by being discovered at conscious times when the organism has high attention, or being connected to other important rules.

The above description prescribes a simple format. This format is one variable strength association between two features. It can encode virtually any database. It also provides a mechanism allowing a format to recode any task in a versatile extensible hierarchical form

fig. 26

A

```
$ lisp
new system now up.  this system will be stable.  (see news)
 2-FEB-1981 21:09:05.72 VAX/VMS LISP of 12-DEC-1980.

? (eval-file 'pam)

 !!! evaluating _DMA1:[TEMP.ANWEJ3]PAM.LSP;1
nil
```


B

```
? (start)
"do you want graphics?"
? nil
"environment file?:"
? maze
"store results t/nil"
? nil
"you have the following functions at your disposal:"

"(crsr-world):  lets user build on *world*"
"(draw-cando):          draws arcs on the umass grinnel"
"(draw-world):          draws all objects in the world"
"(find-rule s1 s2):     nil if not in *cando*"
"(fromhere):            shows connections from/to *last*"
"(sethere):             lets you set here"
"(help):                        prints list of options"
"(make-cando):          lets user build on the *cando*"
"(make-world):          lets user build on the world"
"(clean-rule s1 s2):    makes rules from ones in *cando*"
"(start):               start up system and show all"
"(threshold n)          sets the system threshold to n"
"(travel state):                tries to get to state "
"(viewpoint):           lets user change view"
nil
```

fig. 27

```
? (viewpoint)
"present x land w  mult factors ="
1
1
"input an x multiply factor 1 - 50"
? 2
"input a w multiply factor 1 - 50"
?  3.5

((th (497 470) 1) (six (470 440) 1)
                  (n (440 470) 1)
                  (tw (470 500) 1)
                  (start-box (166 230) 7)
                  (startof-a (156 180) 6)
                  (restof-a (112 192) 6)
                  (seal-box (101 241) 8)
                  (route-c (115 260) 4)
                  (detour-b (182 163) 4))
"input an object in world which you want as refrence"
? tw



? (viewpoint)
"present x land w  mult factors ="
2
 3.5
"input an x multiply factor 1 - 50"
? 1
"input a w multiply factor 1 - 50"
? .5

((th (497 470) 1) (six (470 440) 1)
                  (n (440 470) 1)
                  (tw (470 500) 1)
                  (start-box (166 230) 7)
                  (startof-a (156 180) 6)
                  (restof-a (112 192) 6)
                  (seal-box (101 241) 8)
                  (route-c (115 260) 4)
                  (detour-b (182 163) 4))
"input an object in world which you want as refrence"
? junk
"input xoffset"
? 45
"input woffset"
? 56
56
```

EXAMPLE

To illustrate the internal representation of the pair, consider a simple organism latent learning task, with an initial knowledge base:

PAIR	SYMBOLIC PAIR REPRESENTATION
rest, home	1 2,3
home, rest	4 3,2
pond, drink	5 6,7
field, eat	8 9,10
tree, eat	11 12,10

In the above table, the first element of each pair is the source and the second, the destination. To create a symbolic representation, a unique number is chosen to represent each pair. Each source and destination which is not a pair, is a terminal. These terminals are also given unique "symbolic names". In this example, these are numbers as well. In this case "rest" is represented by 2, "pond, drink" by 5 etc. In this simple example such details as strength of belief and proximity are peripheral and omitted.

By "exploring", the organism has found how to get from home to field, we now get the new node:

home, field 13 3,9

If it is at home and becoming hungry it can look for

home, eat

This pair is found by following eat to find field which con-

nected with home and is represented in the database as:

14 3,8

This rule has as its destination another rule. This is an example of the very most primitive inference which the system can make. By using rule 8 (field, eat) as the destination, an intermediate or process step can be represented in the simple source destination representation. By using a pair as a member of another pair, complex tasks will be encoded in terms of simple tasks. This vertically organized hierarchical structure will form concepts in the same way as the Associatron (Nakano, K. 1972). The system will need an occasional new piece of information to find paths it needs. This new information in a learning system can be hypothesized and tested or simply perceived through the senses.

In this description of PAM, rules are made and can be used to encode other rules. These rules can be used as a production system to encode information. Due to the way that things "learned" at the same time are stored close to each other and the spreading activation of rule search; part of a memory can retrieve a whole memory. The system supports a feature encoded view of the world with the provision for adding any amount of detail. It allows relative beliefs about the relationships between things to change as the overall importance of them increases.

Computer Simulation

A computer program: memrep.lisp has been implemented to demonstrate the graph version of PAM. This program displays endpoints of paths and paths with belief weights intensity encoded on the UMASS Grinnell (Fig. 1). The position of terminals on the screen is chosen to reflect their relative positions in the list: *cando* (Fig. 24), the system's list of rules. *last* is a current location pointer, *cando* is organized topologically to make searches for memories "close" to *last* easy.

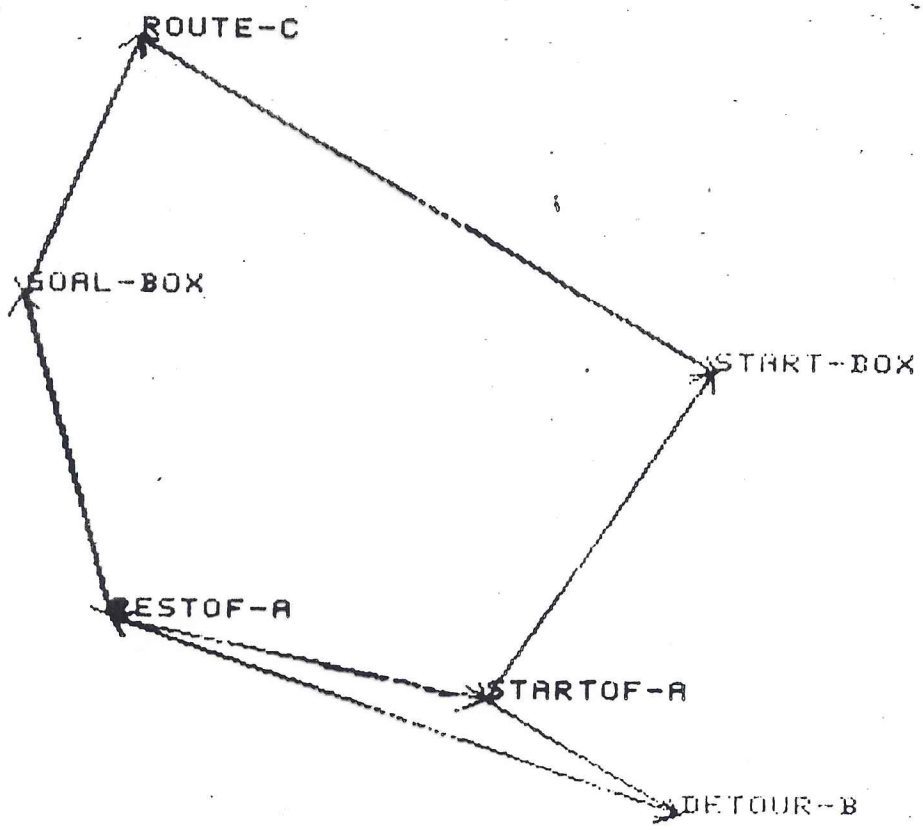
-Sethere (Fig. 25a) allows a user to explicitly set *last*.

- Glean-rule causes a search for a rule specified by a source and a destination. It uses the algorithm already described and adds the rules it discovers to *cando* as it searches. New rules will be made as necessary to find the path. The search can include up to 3 intermediate elements. This number, linguistic experiments have found, is the number of complex associations which a person can manipulate in a sentence (Marcus, M. 1979). If the program user desires, the program will display this search in color raster graphics as it proceeds. The search is a breadth first search from *last* and from the element traveled to.

-Threshold allows a user to change system threshold.

Search time is somewhat independent of how many ele-

fig. 28

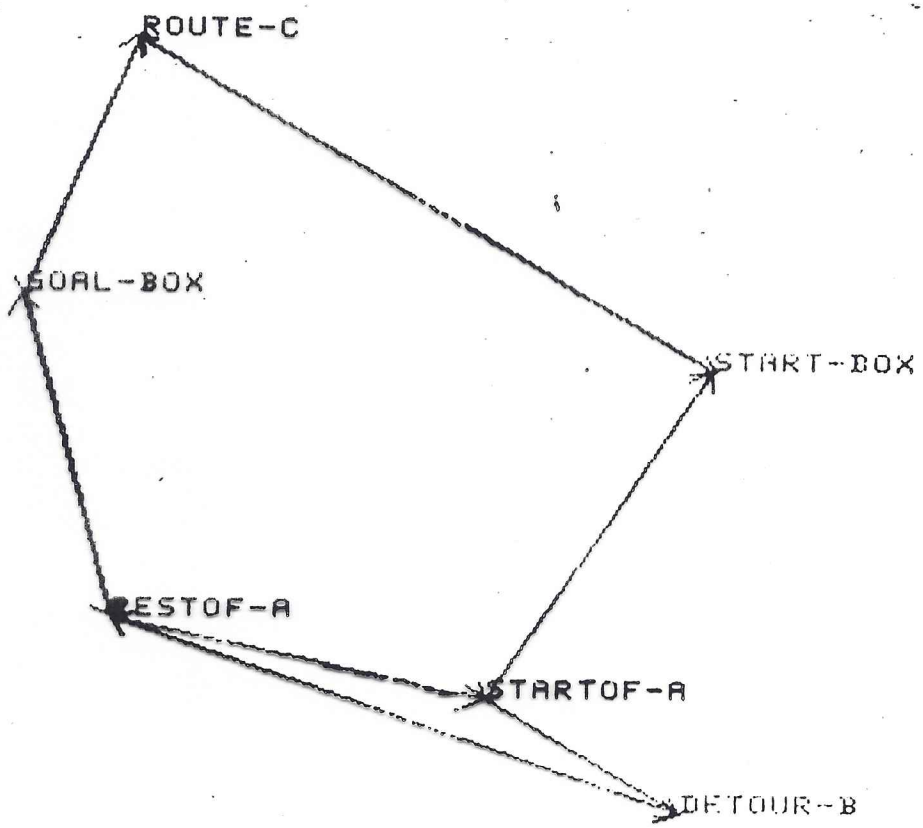


FUNCTION: CANDO
YOU ARE AT: START-BOX

Fig. 29

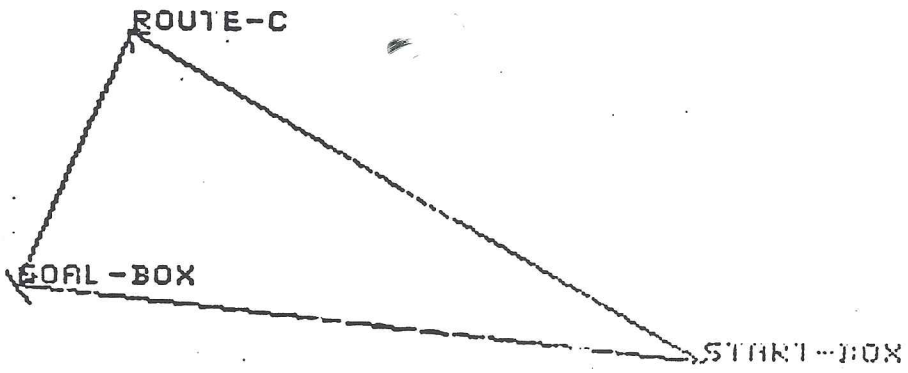
FUNCTION: EKE-RULE
YOU ARE AT: GOAL-BOX

Fig. 28



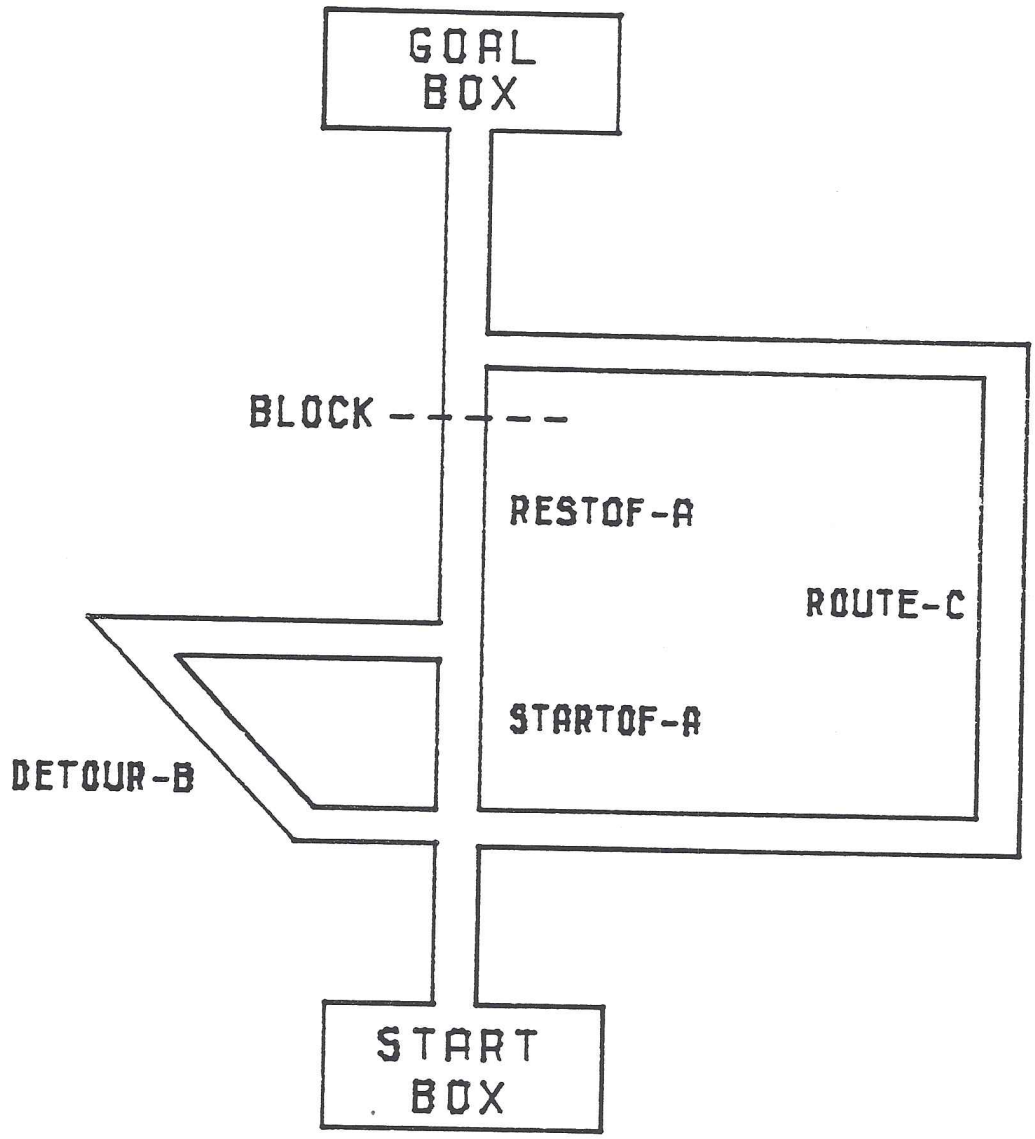

```
FUNCTION:    CANDO
YOU ARE AT: START-BOX
```

fig. 30



FUNCTION: EKE-RULE
YOU ARE AT: START-BOX

fig. 31



ments are in the universe of terminals *world* (the list of terminals) and *cando*, because rules are stored close to rules which beget them. This tends to make rules which are associated in interesting ways be close to each other. The search proceeds starting with rules with *last* in them to simulate spreading activation to improve the search space.

- By using the function Travel with a known element, the system will look for a path to that element using Solve-for-rule.

- By using the function Travel with a new terminal, the system assumes that sensed data is being added, or a hypothesis is being formed, and places the terminal within a probabilistic distance of *last*.

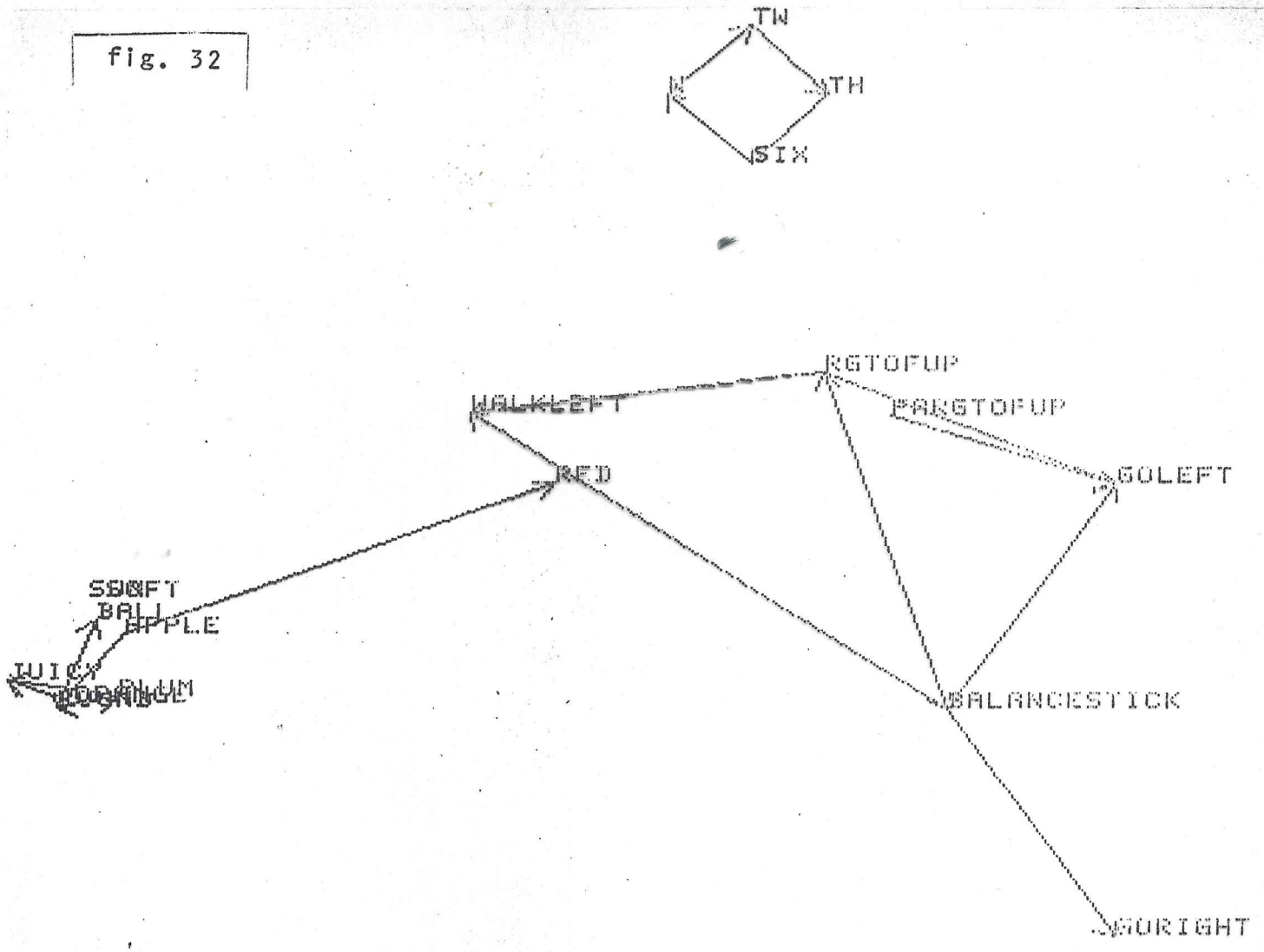
- Start (Fig. 26b) allows a user to bring in a memory file, to display or not display on the Grinnell, and to display the memory's state as viewed from 000,000 at a magnification of 1.

- Viewpoint (Fig. 27a) can be used to change the portion of *world* and *cando* which may be viewed through the Grinnell. The program allows a user to A) specify a terminal element or absolute x,y location as the focal point of the viewport. and to B) specify magnifications for both x and y dimensions for presentation.

- Fromhere displays all paths including *last*. It displays a description of what the system knows near *last*.

- Make-cando (Fig. 24) allows a user to explicitly create

fig. 32



FUNCTION: CANDO
YOU ARE AT: RGTOFUP

fig. 33

GREEN.

SUNSOFT

BALL

APPLE

JUICY

PEACH ROUND ORANGE PLUM

FUNCTION: DRAW-WORLD
YOU ARE AT: RGTOFUP

rules. Such information corresponds to simultaneous and/or important coincident events which an organism learns by experience.

- Make-world (Fig. 25b) allows a user to explicitly create terminals.
- Draw-cando displays all paths in the present window (Grinnell screen).
- Draw-world displays all terminals in the present window.

As a memory system is built, it is stored on disk in a file: the "environment file". Xspread and Yspread define how newly perceived objects (represented by terminals) will be placed on the screen with respect to other objects. This will affect the topology of the memory. For a large environment file using Viewpoint to zoom in and out on the memory structure and pan around, it will be necessary to see the memory structure. The hierarchical nature of path organization is not shown on the display. A natural schema organization develops due to topological placement of rules in *cando*. This structure is a major feature of the Grinnell display. The program listing is Appendix A.

Experiments

Several experimental environments have been constructed using the program for PAM. Two exemplary demonstrations were chosen for this paper. A simple maze environment is used to show how the program simulation of the model can respond in latent learning situations (Fig. 28 and Fig. 29). A small database (Fig. 32) about this paper is the second demonstration. It is used to show how data retrieval occurs in PAM.

Some early experiments with rats developed simple maze problems which test insight or reasoning (Fig. 31)(Tolman, E.C. and Honzik, C.H. 1930). Initially, rats were allowed to become familiar with all portions of Tolman and Honzik's maze. Three possible routes between the Start-Box and the Goal-Box were discovered by the rats: 1. Startof-A Restof-A, 2. Detour-B Restof-A, and 3. Route-C. The rats preferred the routes in inverse order of increasing path length (they prefered 1 better than 2 better than 3). A block was inserted making Restof-A impassible. The rats could have re-explored their possible paths or used an internal map and reasoning to gain insight as to how to get to the goal-box. In this demonstration of so-called latent learning, the rats used their understanding of the maze to solve the problem. The rats knew that Detour-B would be

blocked and took Route-C even though Detour-B was closer.

Tolman and Honzik's maze (Fig. 31) was constructed in PAM using the simulation program (Fig. 28). The topology which the computer simulation created is probabilistic. The features (Start-Box, Startof-A, etc.) are entered as the maze is explored; the program places them close to the associated features. The brightness of a specific connection is encoded in the brightness of the vector which represents it. The Route-A is travelled more often and has been encoded as 3 times the strength of Route-C connections. When the system trys to travel between Start-Box and Goal-Box, it chooses Route-A (Fig. 29). When this is done the system encodes a new feature "Startof-ARestof-A" and another one, "startof-AGoal-Box" to describe this path. This new high level feature, describes the low level description of how to perform the task. When Restof-A is blocked (Fig. 30) the travel command chooses Route-C. In this demonstration PAM performs exactly as other models of latent learning.

If some old information is discovered to be false while attempting to perform a task using it, PAM will either have to find another piece of information to base the task on or eliminate the validity of the task. In the above example the Startof-ARestof-A feature got invalidated by one of its basis, Restof-A being blocked.

The second demonstration that we shall describe is a sample database (Fig. 5 and Fig. 6). For cosmetic reasons the database was entered explicitly defining where things were to be placed. These placements are not very different than what the computer would have chosen itself. The picture on the cover of this paper (Fig. 1) is a figure made by the simulator as it encoded new features to describe the association between Ted and PAM. Using spreading activation (a breadth first search), the simulator found Master-Degree associated with Ted and PAM associated with Project with enough strength to justify building a new association for TedMaster-Degree. To do this, PAM first encodes the source feature TedMaster-Degree which is then associated with PAM. The new association of PAM and Ted is a new primitive that can be used to build future associations.

Figures 2, 3, 4, and 34 are examples of what the simulation can find as the single, above threshold associations that it has for PAM, COINS, UMASS and TED respectively. Fig. 32 and Fig. 33 are included to show how two or more environments can be represented in PAM. VIEWPOINT is used in these two figures to examine the environment. These figures show how using the functions SETHERE, and FROMHERE a user can fly around a database searching for related things. To see what is in the database one can use VIEWPOINT and DRAW-WORLD as in Figs. 6, 28, 32, 33 . All associations in

the database are shown in figure 6 using DRAW-CANDO.

This database demonstration shows how PAM could be used to enter and retrieve data. Using PAM as a high level tool one can store and retrieve data from multiple knowledge sources in seperate schemata (Fig. 32 and Fig. 33). In such a representation serial associations and hierarchies of associations structure the data. If primitive features are to encode real world defined things they can drive Input and/or Output routines.

Limitations and Future Work

Two major tests which PAM has not been given are: Closed loop reinforcement in an environment and implementing a simulation of the distributed definition of PAM.

Threshold for accepting associations, and rules for how much to change an association's believability when travelled, could be controlled adaptively. As the organism's memory search gets stuck in an unknown situation it should lower its threshold. Its threshold should also be changed depending by what the are reasons for searching the memory. For example the reasons for searching memory change as an organism's behaviorial mode changes from fight to flight to sleep etc.. These mode changes will of course be affected by many different influences. Such diverse structures as the primitive Reticular Formation and the brain's most recent acquisition, the Cerebral Cortex, have dramatic affect on aspects of interest, attention, focus and other influences on memory and retrieval abilities. For this reason these many structures influence closed loop organism learning.

However, even without adding other structures to PAM, it can encode any algorithmic decision process for strategy control. Using external sensory feedback heuristics could

be encoded as algorithms which are controlled by the environment. Simulations of the adaptive strategies which control some of the closed loop learning would be useful. Such adaptive elements as ASN (Barto, A., Sutton, R., and Brower, P. 1980) could be used to contol memory threshold for learning and retrieval.

For this to be useful, a hardware interface to an environment or a simulated world-like environment which can give feedback are required.

A simulation of the neural network with external control of acceptance threshold, learning constant, and lateral inhibition level would be an extremely interesting testbed for the model.

Conclusion

The model described in this paper, PAM, has two important points to share. 1. Semantic networks do not need labeled edges. A simplified semantic network representation using the bare minimum of committed definitions can encode what classical semantic networks store. Such a scheme fits perceptual data better than previous alternatives described in the net formalism. Unlike many other systems, the network definition does not need to be modified to encode new knowledge.

2. A relationship between high level (semantic network and production system) definitions and low level (or neural) descriptions, is both possible and surprisingly easy. By recoding the language of a model, people from different disciplines can evaluate and share their ideas.

Just as two states encode any number in the binary number system, the combination of simple associations between two things can encode any association.

Any knowledge can be encoded in PAM. For example: simple decision making can be encoded in hierarchy and sequences of rules; weighting can be used to encode alternative solutions. heuristics can be encoded using the same

technique.

PAM structures memory: encoding similar things close to each other. Similarity, is defined by time proximity of when things were learned. Similarity is increased between things as they get used together. This topological memory building decreases search time for "usual" situations.

BIBLIOGRAPHY

Anderson, J.R., and Bower, F. H.: Human Associative Memory. Washington, D.C.:V.H. Winston & Sons 1973

Albus, J.S.:Mechanisms of Planning and Problem Solving in The Brain. Mathematical Biosciences 45:247-293 (1979)

Barto, A. G., Sutton, R. S., Brouwer, P.: Associative search network: A reinforcement learning associative memory. To appear in: Biol. Cybernetics (1981)

Berliner, H.J.: Backgammon Computer Program Beats World Champion. Artificial Intelligence 14,2 (1980)

Brinton, J.B.: Wafers to Challenge Disks, Bubbles. Electronics Aug 16 1979

Brooks, R.S.: A Hardware Semantic Net. Unpublished. University of Massachusetts 1975

Brown, R.W., and Mcneil, D.: The Tip of The Tongue Phenomenon. Journal of Verbal Learning and Verbal behavior 5,325-337 (1966)

Carpenter, M.B.: Human Neuroanatomy Seventh Edition Williams & Wilkins, Baltimore Md. 1976

Chiang, C.: A model of a Human Recognition System with "Thinking". Biol. Cybernetics 36, 131-135 (1980)

Cooper, L.A., and Shepard, R.N.: Chronometric Studies of the Rotation of Mental Images. In W.G. Chase (Ed.), Visual Information Processing NY: Academic Press 1973

Daniels, J.: M-Cells: A Logic Ciruit Model to Account for Some Features of CNS Inhibition. Biol. Cybernetics, 1-9 (1978)

Duda, R.O., and Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, NY 1973

Eccles, J.C.: The Understanding of the BRAIN. McGraw-Hill 1977

Eimas, P.D., and Corbit, J.D.: Selective Adaptation of Linguistic Feature Detectors. Cognitive Psychology vol 4 (1973)

Erman, L.D., Lessar, V.R.: A Multi-level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge, Advance Papers of the Fourth International Joint Conference on Artificial Intelligence (1975)

Fahlman, S.E.: A System for Representing and Using Real World Knowledge. MIT Artificial Intelligence Lab Memo 331 1974

Fahlman, S.E.: NETL A System for Representing and Using Real World Knowledge. MIT Press, Cambridge, Mass 1979

Foster, D.H.: A Description of Discrete Internal Representation Schemes for Visual Pattern Discrimination. Biol. Cybernetics 38, 151-157 (1980)

Fox, C.A.: Correlative Anatomy of the Nervous System. Macmillan 1962

Fukushima, K.: Neocognitron: A Self-organizing Neural Network model for a mechanism of Pattern Recognition Unaffected by Shift in position. Biol. Cybernetics 36, 193-202 (1980)

Hebb, D.O.: Organization of behavior. Wiley 1949

Hebb, D.O.: Intelligence in Man after Large Removal of Cerebral Tissue: Report of Four Left Lobe Cases. J. General Psychology 21, 73-87 (1939)

Hubel, D.H., and Wiesel, T.N.: Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. Journal of physiology 160, 106-154 (1962)

Jackson, J.: Selected Writings of John Hughlings Jackson. Hodder and Stoughton, London 1931

Katz, J.J., and Fodor, J.A. The structure of a semantic theory. language, (1963)

Klatzky, R.L.: Human Memory Structures and Processes. Freeman and Company 1975

Kohonen, T.: Associative memory: A system theoretic approach. Berlin: Springer-verlag 1977

Kung, H.T.: The Structure of Parallel Algorithms. Dept. of Computer Science, Carnegie Mellon University. 1979

Larkin, J., McDermott, D.P., Simon, H.A.: Expert and Novice Performance in Solving Physics Problems. Science Vol 208. 20 (1980)

Lashley, K.S.: Brain Mechanisms and Intelligence: A Quantitative Study of Injuries to the Brain. Univ. Chicago Press 1929

Lettvin, J.Y., Maturana, H., McCulloch, W.S., and Pitts, W.H.: What the Frog's Eye Tells the Brain. Proc. IRE 47, 1940-1951 (1959)

Kuipers, B.: modeling Spatial Knowledge. Cognitive Science 2, 129-153 (1978)

Lynch, K. The Image of the City. MIT Press, Cambridge, Ma 1960

McDermott, D., and Doyle, J.: Non Monotonic Logic. Artificial Intelligence 13,1 (1980)

Maclaren, L.S.: A Production System Architecture Based on Biological Examples. Ph.D. Dissertation, Department of Computer Science. University of Washington 1980

Marcus, M.: The Theory of Syntactic Recognition. PHD Disertation MIT AI Lab 1977

Marr, D.: A Theory for Cerebral Neocortex. Proc. Roy. Soc. Lond. B 176, 161-234 (1970)

Marr, D., and Nishishihara, H.K.: Visual Information Processing; Artificial Intelligence and the Sensorium of Sight. Technology Review Vol 81, 1 (1978)

Marr, D., and Hilderith, E.: theory of Edge Detection. MIT Memo AIMEMO 518 (1979)

Michie, D., and Chambers, R.A.: BOXES: An Experiment in Adaptive Control. Machine Intelligence. Editors; Dale, E., and Michie, D.: Oliver and Boyd; Edinburgh 1968

Miller, G.A. English verbs of motion: A case study in semantics and lexical memory. In A.W.Melton and E.Martin (Eds.), Coding Processes in Human Memory. Washington, D.C.: V.H. Winston & Sons 1972

Millward, R.: model of Concept Formation. Center For Cognitive Science. Brown University 1978

Minsky, M.: the Psychology of Computer Vision. Winston, P.: Editor. McGraw Hill Inc. NY 1975

Minsky, M.,Papert, S.: Perceptrons An Introduction to Computational Geometry. The MIT Press Massachusetts Institute of Technology 1969

Minsky, M.: Semantic Information Processing. The MIT Press 1968

Moreno-Diaz, R.,Rubio, E.: A model For Non-linear Processing in Cat's Retina. Biol. Cybernetics 37,1 25-32 (1980)

Nakano, K.: Associatron: A model of Associative Memory. IEEE Transactions on Systems, Man, and Cybernetics. 3, 380-387 (1972)

Newell, A., Shaw, J.C., and Simon, H.A.: A Variety of Intelligent Learning in a General Problem Solver. in Yovitis, M., and Cameron, S. (eds.), Self-Organizing Systems, Pergamon, NY 1960

Newell, A., and Simon, J.C.; Computers in Psychology, in Luce, R.D., Bush, R., and Galanter, E. (eds.), Handbook of Mathematical Psychology Vol 1, Wiley, NY 1963

Nilsson, N.J.: Problem Solving Methods in Artificial Intelligence. McGraw-Hill, NY 1971 p; Osgood, C.E. The nature and measurement of meaning. Psychological Bulletin 49, 197-237 (1952)

Palm, G.: On Associative Memory. Biol. Cybernetics 36, 19-31 (1980)

Penfield, W., and Rasmussen, T.: The Cerebral Cortex of Man. A Clinical Study of Localization of Function. Macmillan Company, NY 1950

Piaget, J.: Origins of Intelligence in Children. International Press, Cook, Trans., Norton 1963

Posner, M.I., and Keele, S.W.: On the genesis of abstract ideas. Journal of Experimental Psychology 73, 28-38 (1968)

Quillian, M.R.: Semantic Memory. Semantic Information Processing. The MIT Press Cambridge Ma 1968

Quillian, M.R.: The Teachable Language Comprehender: A Simulation Program and Theory of Language. Communications of the Association for Computing Machinery 12, 459-476 (1969)

Rips, L.J., Shoben, E.J., and Smith, E.E.: Semantic Distance and the Verification of Semantic Relations. Journal of Verbal Learning and Verbal behavior 12, 1-20 (1973)

Rychener, M.D.: Production Systems as a Programming Language for Artificial Intelligence. Ph.D. Dissertation, Carnegie-Mellon University Department of Computer Science, Pittsburgh, Pa 1976

Selfridge, O.G.: Pandemonium: A Paradigm for learning. In The mechanization of Thought Processes. London H.M. stationary Office 1959

Selker, E.J.: An Image Based Focusing System. Undergraduate honors Thesis, Brown University 1979

Skinner, B.F.: Beyond Freedom and Dignity. Random House, NY 1971

Sherwin, I.: Temporal Lobe Epilepsy: Neurological and behavioral Aspects. Annual Rev. of Medicine 37 (1976)

Spinelli, D.N.: OCCAM: A Computer model For A Content Addressable Memory in The Central Nervous System. Biology of Memory. Academic Press, NY 1970

Thomson, D.M., and Tulving, E.: Associative encoding and retrieval: Weak and Strong Cues. Journal of Experimental Psychology 86, 255-262 (1970)

Tolman, E.C. and Honzik, C.H.: "Insight" In Rats. Univ. Calif Publ. Psychol., 4, 215-232 (1930)

Widrow, B., Gupta, N.K., and Maitra, S.: Punish/Reward: Learning with a a Critic in Adaptive Threshold Systems. IEEE Transactions on Systems, Man, and Cybernetics 3, 5, (1973)

fig. 34


```
FUNCTION:   FROMHERE
YOU ARE AT: TED
```

APPENDIX

```
(setq *lower-lock t)
(defun start ()
(comment ***********************************************************
* this program implements the memory representation formalism put    *
* forward in memrep.doc.                                             *
* all objects in the models universe are in *world*,                 *
* all infrmation  has about thier interdependancies in *cando*.      *
* these two lists live in learnks.lsp and are updated by the program *)
(comment ***********************************************************
* this is the initialization of the program; it must be called *
* at the beggining of each session.                            *
***********************************************************)
(prog ()
(print "do you want graphics?")
(setq grin (read))
    (cond (grin(gr_initialize 0 nil ))
              (t (gr_initialize 1 "dma1:[temp.anwejsljunk.dat")))

(print "environment file?:")
(setq iofile (read))

(print "store results t/nil")
(setq *store (read))

(cond ((not iofile) (setq iofile 'blank.lsp)))
    (gr_config444)
(gr_add_back)

(comment **initialize global variables**)
    (load-file iofile)

(comment **variables for placing new terminals**)
    (setq land and)
    (setq lor or)
    (setq *xspread 100)
    (setq *hlfxspread (quotient *xspread 2))
    (setq *yspread 100)
    (setq *hlfyspread (quotient *yspread 2))

(comment *importance pairs need to be noted in search)
    (setq *thresh 0 thresh 0)

(comment **viewport parameters**)
    (setq *xzoom 1)
    (setq *yzoom 1)
    (setq *xoffset 0)
    (setq *yoffset 0)
    (setq trackflg t)       (comment track entered things as *last)

(comment **loop to set up random number generater)
(setq rnd *randcount)
setrand (random *yspread)
    (setq rnd (sub1 rnd))
    (cond ((greaterp rnd 0)(go setrand)))

    (cond ((not(find *last *cando*))(make-searchlist *cando*))
            (t (pp *cando*)
            (print "input desired *last from above list")
            (setq *last (read))))
            (draw-world)
            (gr_text 230 20 *last 0 1 1 0 0 0 255)
            (typefunc 'start)
            (help)))
```

```
(defun help ()
(comment ***************************************************************
* this function prints user available function names                   *
***************************************************************)
(typefunc  'help)
    (print "you have the following functions at your disposal:" )
    (terpri)
    (print "(crsr-world):    lets user build on *world*")
    (print "(draw-cando):           draws arcs on the umass grinnel")
    (print "(draw-world):           draws all objects in the world")
    (print "(find-rule s1 s2):      nil if not in *cando*")
    (print "(fromhere):             shows connections from/to *last")
    (print "(sethere):              lets you set here")
    (print "(help):                 prints list of options")
    (print "(make-cando):           lets user build on the *cando*")
    (print "(make-world):           lets user build on the world")
    (print "(glean-rule s1 s2):     makes rules from ones in *cando*")
    (print "(start):                start up system and show all")
    (print "(threshold n)           sets the system threshold to n")
    (print "(travel state):         trys to get to state ")
    (print "(viewpoint):            lets user change view")nil)
```

```
[defun ablate (action conseq)
(comment ***************************************************************
* this routine allows a user to get rid of rules                        *
***************************************************************)

(cond ((setq rule (find-rule action conseq))
    (remove-element rule *cando*)]


(defun draw-cando ()
(comment  ***************************************************************
* this function draws lines between all objects in the world which*
* are connected (umass grinnell), color encodes importance        *
***************************************************************)
(typefunc 'cando)
    (mapc *cando*
            '(lambda (thing)
                    (set (car thing) (car thing))
                    (dracando thing)
                    )))



(defun draw-world ()
(comment ***************************************************************
*  this function puts all names of things in the world at their   *
*  possition on the umass grinnell, intensity encodes connectivity*
***************************************************************)
    (gr_clear)
    (GR_RECT 0 0 511 511 255 255 255)
    (drawstats)
(typefunc 'draw-world)

    (mapc *world*
      '(lambda (thing)
            (set (car thing) (car thing))
            (drawrld (car thing))
            )))

(defun find-rule (state1 state2)
(comment ***************************************************************
* this function simply checks to see if a state pair is in *cando* *
***************************************************************)

    (setq itt nil)
    (mapc *cando*
      '(lambda (sd)
            (cond((land (eq (left-most sd) state1)
                        (eq (right-most sd) state2)
                        (greaterp (ival sd) *thresh))
              (setq itt sd) (dracando sd))
            )))itt)
```

```
(defun make-world ()
(comment ***********************************************************
*   this function alows you to enlarge the *world* data base      *
* this data base is in the form (thing1(x y))(thing2(x y)...)     *
* it prompts user for successive entries. it exits on entry = nil*
***********************************************************)
(typefunc 'make-world)
    (prog (it)
    (pprint *world*)
    (print "if you want rules made enter r")
    (setq r (read))
    (print "if you have any additions; type them in with format")
      (print "(thing (x y)z) where z is current # connections")
      (print "otherwise type nil")

    (comment **do while new entries**)
rdworld     (setq it (read))
    (cond (it( setq *world* (append *world* (list it)))
    (cond (r(plod *last  (car it))(setq *last (car it))))

            (storenviron)
            (gr_text (x-val it)(y-val it) (car it) 0 1 1 0
            (difference 255 (ival object)) (difference 255 (ival it)) 255)
            (gr_flush)
            (go rdworld)))))



[defun make-cando ()
(comment ***********************************************************
*   this function allows you to enlarge the *cando* knowlege      *
*  let your system explore: the data base is the form:            *
*  (cando (label (state1 state2)))                                *
* it prompts user for successive entries. newrle exnewrles on entry = nil*
***********************************************************)
(typefunc 'make-cando)
    (prog ()
    (pprint *cando*)
    (print "if you have any additions to *cando*; put in form:")
    (print "(label (state1 state2) z) where z is importance")
    (print "type nil when done with entries")

    (comment ** do while new entries**)
rdcando     (setq newrle (read))
(cond
    ((not (find (print (left-most newrle)) *world*))
    (print "not in *world*: place?nil/y")

(comment ****make new thing in world****)
    (cond ((read)(plant (left-most newrle)))
            (t(go rdcando))))
    (t(cond((not (find (print (right-most newrle)) *world*))
            (print "not in *world*: place? nil/y")
            (cond ((read)(plant(right-most newrle)))
                    (t(go rdcando)))))
    (cond (newrle( addtocando  newrle)
            (setq *last (right-most newrle))
            (go rdcando)]
```

```
(defun crsr-world ()
(comment ***************************************************************
*   this function alows you to enlarge the *world* data base      *
* this data base is in the form (thing1(x y))(thing2(x y)...)     *
* it uses the crsr to get inputs. It exits on entry = nil         *
***************************************************************)
(typefunc 'crsr-world)
    (prog (it)
    (pprint *world*)
    (print "if you want rules made enter r")
    (setq r (read))
    (print "if you have any additions;")

    (comment **do while new entries**)
rdworld(print "nil or new *last")
    (cond ((setq dummy (read)) (setq *last dummy)))
    (print "nil or new thing")
            (setq it (read))
    (cond (it( setq *world* (append *world* (list(setq it(list it
                              (gr_crsr_read_wt 1)0)))))
            (gr_text (x-val it)(y-val it) (car it) 0 1 1 0
            (difference 255 (ival object)) (difference 255 (ival it)) 255)
            (gr_flush)
    (cond (r(plod *last (car it))))

            (go rdworld)))))



[defun viewpoint ()
(comment ***************************************************************
* this function allows a user to center the grinell viewport about  *
* an object                                                         *
***************************************************************)
(typefunc 'viewpoint)
    (print "present x land y  mult factors =")
    (print *xzoom )(print *yzoom)
    (print "input an x multiply factor 1 - 50")
    (setq *xzoom (read))
    (print "input a y multiply factor 1 - 50")
    (setq *yzoom (read))
(pp *world*)
(print "input an object in world which you want as refrence")
(cond ((not(setq it (find (read) *world*)))
    (print "input xoffset")
    (setq *xoffset (read))
    (print "input yoffset")
    (setq *yoffset (read)))
      (t(setq *xoffset (plus (minus (x-val it)) (quotient 128 *xzoom)))
    (setq *yoffset (plus (minus (y-val it))(quotient 128 *yzoom)]
```

```
[defun remove-ival (action conseq)
(comment ***************************************************************
* this function allows a user to temporarily remove a rule's ival *
***************************************************************)
(typefunc 'remove-ival)
(cond ((setq rule (find-rule action conseq))
    (nconc (list (car rule)) (list (cadr rule)) '(0) (cddr rule)]


[defun restore-ival (action conseq)
(comment ***************************************************************
* this function allows a user to restore a rules ival           *
***************************************************************)
(typefunc'restore-ival)
(cond ((setq rule (find-rule action conseq))
    (remove-element 0 rule]


[defun fromhere ()
(comment ***************************************************************
* this function shows what connecttions are possible from *last  *
***************************************************************)
(GR_RECT 0 0 511 511 255 255 255)
(drawstats)
(typefunc 'fromhere)
(gr_flush)
(mapc *searchlist
    '(lambda (qhere)
            (cond ((lor (eq (left-most qhere) *last)
                    (eq (right-most qhere) *last))
                    (drawrld (left-most qhere))
                    (drawrld (right-most qhere))
                    (dracando qhere)]


(defun travel (spot)
(comment ***************************************************************
* this macro simply calls glean-rule to try to go somewhere     *
***************************************************************)
(typefunc 'travel)
(cond ((lor (find spot *world*)(find spot *cando*))
     (glean-rule *last spot)
    (print "input nil if you do not want to try to make a track")
    (cond ((read)(plod *last spot)(setq *last spot))))

     (t (plant spot))))
```

```
[defun plant (spot)
(comment ************************************************************
* this function places objects at a random place within a radius     *
* *xspread by *yspread to *last in the *world*                       *
************************************************************)
(typefunc 'plant)

(setq last (find *last  *world*))

    (comment **place newly found object in *world* make a place)
     (comment **not to near either side of world)

    (comment **if to close to left edge of screen)
     (cond ((greaterp *hlfxspread (setq xval (x-val last) ))
            (setq xval *hlfxspread))

            (nil  (comment **else if to close to right edge))
            ((greaterp  xval  (difference 511 *xspread))
              (setq xval (difference xval *xspread)))
              (setq xval (difference xval *hlfxspread)))

      (comment **not to near top or  bottom of world**)
      (comment **if not to close to bottom of screen)
      (cond ((greaterp  *hlfyspread (setq yval (y-val last)) )
             (setq yval *hlfyspread))

       (nil (comment **if not to close to top of screen))
             ((greaterp yval (difference 511 *yspread))
              (setq yval(difference yval *yspread)))
              (setq yval (difference yval *hlfyspread )))

       (comment **place it near where we are**)
      (setq *randcount (plus *randcount 2))
       (addtoworld (list spot
            (list (fix (plus (random *xspread) xval
                             (minus *hlfxspread)))
                  (fix(plus (random *yspread) yval (minus *hlfyspread))))
                      0))
    (print "input nil if you do not want to try to make a track")
    (cond ((read)(plod *last spot)(setq *last spot)))]
```

```
(defun eke-rule (state1 state2)
(comment  ***********************************************************
* this function is the heart of the associating process, it can *
* create a new pair entry in *cando* if there is a possible     *
* way to traverse two states using up to one intermediate pair  *
***********************************************************************)
(typefunc 'eke-rule)
 (setq eke t)
 [cond (( not (up-ival(find-rule state1 state2)))

 (comment **look for a leftbase: a pair with the state1 as its action**)
 (setq eke nil)
 (mapc *searchlist
   '(lambda (leftbase)

    (comment **if action side of leftbase fits then**)
    (cond ((land(eq (left-most leftbase) state1) (not eke)
            (greaterp (ival leftbase) *thresh))
            (drawrld (right-most leftbase))
            (dracando leftbase)

            (comment **look for a rightbase with the other half**)
            [mapc *searchlist
            '(lambda (rightbase)
               (cond [(land(eq (right-most rightbase) state2)(not eke)
                           (greaterp (ival rightbase) *thresh))
                      (drawrld (left-most rightbase))
                      (dracando rightbase)

                  (comment **if rule can be made then**)
                  (cond[(eq(right-most leftbase)
                           (left-most rightbase))

                    (comment **make it**)
                    (up-ival rightbase)
                    (up-ival leftbase)
                    (setq eke t)
                    (addtocando (list(pack
                      (list  (car leftbase) (car rightbase)))
                      (list (car leftbase) (car rightbase) )
                                          (plus thresh 1)]

                  (t(undracando rightbase)
                  (undrawrld (left-most rightbase))]]
            (cond ((not eke)
                  (undracando leftbase)
                  (undrawrld (right-most leftbase)]
    eke)
```

```
(defun glean-rule (action conseq)
(comment ************************************************************
* this function uses eke-rule to look for rule connections      *
* up to a depth of 4 rules using a modified bredth first search*
************************************************************)
(GR_RECT 0 0 511 511 255 255 255)
(typefunc 'glean-rule)
(drawstats)
(drawrld action)
(drawrld conseq)
(make-searchlist *cando*)

(comment **if not a simple intermediate**)
[cond ( (eke-rule action conseq)
    (setq solve t)
    (setq *last conseq))
    [t (setq solve nil)

    (comment **then look for possible secondary intermediates**)
    (mapc *searchlist
    '(lambda (newlft)
            (cond ((not solve)
                   (cond ((land(eq (left-most newlft) action)
                               (greaterp (ival newlft) *thresh))
                          (drawrld (right-most newlft))

                     (cond [( not(mapc *searchlist
                     '(lambda (newrgt)
                           (cond ((not solve)
                             (cond((land(eq(right-most newrgt)conseq)
                               (greaterp(ival newrgt) *thresh))
                               (drawrld (left-most newrgt))

                               (cond[(eke-rule (right-most newlft)
                                               (left-most newrgt))
                                     (eke-rule action
                                               (left-most newrgt))
                                     (setq *last conseq)
                                     (eke-rule action conseq)
                                     (setq solve t)]
                                    (t (undrawrld
                                               (left-most newrgt))]
                                (t(undrawrld (right-most newlft]]
solve)
```

```
(defun left-most (object)
(comment ***************************************************************
*this macro finds the terminal which represents the start point     *
*for a pair recursively.                                            *
***************************************************************)
    (cond ((not (listp object)))
          ((not (find (caadr object) *world*))
             (cond ((setq nxtobject  (find (caadr object) *cando*))
                    (left-most nxtobject))))
           (t(caadr object))))


(defun right-most (object)
(comment ***************************************************************
*this macro finds the terminal which represents the  end point      *
*for a pair recursively.                                            *
***************************************************************)
    (cond ((not (listp object)))
          ((not (find (cadadr object) *world*))
             (cond ((setq nxtobject (find (cadadr object) *cando*))
                    (right-most nxtobject))))
           (t(cadadr object))))




(defun xdiff ( thing2 thing1)
(comment ***************************************************************
* this macro takes the differences of the x-val of thing2       *
* and thing1                                                    *
***************************************************************)
    (fix(difference (times *xzoom (plus (x-val thing2) *xoffset))
                    (times *xzoom (plus (x-val thing1) *xoffset)))))


(defun ydiff (thing2 thing1)
(comment ***************************************************************
* this macro takes the differences of the y-val of thing2       *
* thing1                                                        *
***************************************************************)
    (fix(difference (times *yzoom (plus (y-val thing2) *yoffset))
                    (times *yzoom (plus (y-val thing1) *yoffset)))))
```

```
(defun x-val (object)
(comment ***************************************************************
*  this macro gets the value of an object in the *world* list  *
***************************************************************)
    (caadr object))


(defun y-val (object)
(comment ***************************************************************
*   this macro gets the value of an object in the *world* list *
***************************************************************)
    (cadadr object))


(defun ival (object)
(comment ***************************************************************
* this macro finds the value of the importance level for        *
* objects in the world, and metric of good or bad for pairs     *
***************************************************************)
(plus (fix (times (caddr object)  (quotient 200 *maxival))) 50))



[defun up-ival (object)
(comment ***************************************************************
*   this macro increments the ival on any pair sent it            *
***************************************************************)
(cond [object
    (rplaca (cddr object) (add1 (caddr object)))
    (cond ((greaterp (caddr object) *maxival)
           (setq *maxival (caddr object)))
           (t)])]


[defun dwn-ival (object)
(comment ***************************************************************
*                                                                     *
*   this macro decrements the ival on any pair sent it            *
***************************************************************)
(cond [object
    (rplaca (cddr object) (sub1 (caddr object)))
    (cond ((greaterp (caddr object) *maxival)
           (setq *maxival (caddr object)))
           (t)])]
```

```
(defun find ( fst lst)
(comment ***************************************************************
*  this macro looks through a lst for an element fst as the car *
* of an element of the lst it returns that element                *
***************************************************************)
    (setq it nil)
    (mapc lst
            '(lambda thing
                    (cond ((eq fst (caar thing))
                    (setq it (car thing)) ))))it)
(defun plod (here there)
(comment ***************************************************************
* this function puts in a connection between here and there iff   *
* the x distance is not farther than *xspread and the y distance  *
* is not farther than *yspread.                                   *
***************************************************************)
(cond ((land(greaterp *xspread
    (abs:i (difference (x-val (find there *world*))
                (x-val (find here *world*)))))
    (greaterp *yspread
    (abs:i (difference (y-val (find there *world*))
                (y-val (find here *world*))))))
    (addtocando (list (pack (list here there))(list here there)
 (plus thresh 1)))
    (cond (trackflg (setq *last there))))))

(defun sethere ()
(comment ***************************************************
*    This macro sets *last                                *
***************************************************)
(print *last)
(print "is presently where you are")
(pp *world*)
(print "input where you want to be")
(setq *last (read))
)


(defun threshold (thresh)
(comment ***************************************************************
* this function sets the system theshold to thresh                 *
***************************************************************)
(setq thresh thresh)
(setq *thresh (plus (fix (times thresh (quotient 200 *maxival))) 50)))
```

```
[defun dracando (object)
(comment ***************************************************************
* this macro simply draws the cando vector for object                  *
***************************************************************)
(setq object1 (find (left-most object) *world*))
(setq object2 (find (right-most object) *world*))

(cond ((clip object2)
    (cond ((clip object1)

(gr_vect zoomedx zoomedy
    (setq dx(xdiff object2 object1))
    (setq dy(ydiff object2 object1))
    (difference 255 (ival object)) 220 (difference 255 (ival object)))

(comment rotate and scale edge to make arrow head tan point angle =.7)

(setq pntsize (quotient 8(plus (abs:i dx) (abs:i dy))))
(setq xleg (plus 1 (fix (times pntsize (difference (times (minus dy) .7)
                                                   (minus dx))))))
(setq yleg (plus 1 (fix (times pntsize (difference (times dx .7)
                                                   (minus dy))))))
(gr_vect_clip (plus zoomedx dx)(plus zoomedy dy)(minus xleg)(minus  yleg)
    (difference 255 (ival object)) 220 (difference 255 (ival object)))
(setq xleg (plus 1 (fix (times pntsize (difference (times dy .7)
                                                   (minus dx))))))
(setq yleg (plus 1 (fix (times pntsize (difference (times (minus dx) .7)
                                                   (minus dy))))))
(gr_vect_clip(plus zoomedx dx)(plus zoomedy dy)(minus xleg)(minus  yleg)
    (difference 255 (ival object)) 220 (difference 255 (ival object)))
(gr_flush))]


[defun undracando (object)
(comment ***************************************************************
* this macro erases a line representing a member of *cando*            *
***************************************************************)
(setq object1 (find (left-most object) *world*))
(setq object2 (find (right-most object) *world*))
(cond ((clip object2)
    (cond ((clip object1)

(gr_vect zoomedx zoomedy (setq dx(xdiff object2 object1))
    (setq dy(ydiff object2 object1)) 255 255 255)

(setq pntsize (quotient 8(plus (abs:i dx) (abs:i dy)) ))
(setq xleg (plus 1 (fix (times pntsize (difference (times (minus dy) .7)
                                                   (minus dx))))))
(setq yleg (plus 1 (fix (times pntsize (difference (times dx .7)
                                                   (minus dy))))))
(gr_vect_clip (plus zoomedx dx) (plus zoomedy dy) xleg yleg
                    255 255 255)

(setq xleg (plus 1 (fix (times pntsize (difference (times dy .7)
                                                   (minus dx))))))
(setq yleg (plus 1 (fix (times pntsize (difference (times (minus dx) .7)
                                                   (minus dy))))))
(gr_vect_clip (plus zoomedx dx)(plus zoomedy dy)  (minus xleg)(minus yleg)
                                    255 255 255)
(gr_flush))]
```

```
[defun drawrld (object)
(comment ***************************************************************
* this macro simply draws a single thing in the world after find- *
* ing it in *world*                                                    *
***************************************************************)
(setq object (find object *world*))
(cond((clip object)

(gr_text zoomedx zoomedy(car object) 0 1 1 0
          (difference 255 (ival object)) (difference 255 (ival object)) 255)
(gr_flush))]


[defun undrawrld (object)
(comment ***************************************************************
* this macro removes a drawn object in *world* from the grinnell *
***************************************************************)
(cond ((setq object (find object *world*))
    (cond((clip object)

    (gr_text (x-val object)(y-val object)(car object) 0 1 1 0
    255 255 255)
    (gr_flush))))]

[defun clip (thing)
(comment ***************************************************************
* this function returns t if object is within 0 -511 in x and y  *
***************************************************************)

(cond ((greaterp 511 (setq zoomedx (fix(times *xzoom
                    (plus (x-val thing) *xoffset)))) 0)
      (cond ((greaterp 511 (setq zoomedy (fix(times *yzoom
                          (plus (y-val thing) *yoffset)))) 0)]



[defun make-searchlist (srclist)
(comment ***************************************************************
* this macro creates a searchlist of *cando* with the pair which     *
* has *last its first element first in the srclist                    *
***************************************************************)
(comment ** recurse through srclist looking for *last as car of pair)
[cond ((atom srclist) nil)
      ((eq *last (x-val (car srclist)))(setq *searchlist  srclist))
    (t  (make-searchlist (cdr srclist))

    (comment ** add pairs occuring before the one with *last
       as its first element at end in reverse order)
(nconc *searchlist (list(car srclist]]
```

```
[defun addtocando (thing)
(comment ***************************************************
*   this routine adds a rule to cando, updates the learnerks *
* file and draws it as a vector.                              *
***************************************************************)
(cond ((land (not (find-rule (x-val thing) (y-val thing)))
     (up-ival (find (right-most thing) *world*))
     (up-ival (find (left-most thing) *world*)))
     (set (car thing) (car thing))
     (rplacd *searchlist (cons  thing (cdr *searchlist)))
             (storenviron)
             (dracando thing)]

[defun addtoworld (thing)
(comment ***************************************************************
* this function simply adds thing to world if it is not already       *
***************************************************************)
(typefunc 'addtoworld)
(cond ((not (find (car thing) *world*))
     (setq *world* (append *world* (list thing) ))
     (drawrld (car thing))
     (storenviron)]



(defun storenviron ()
(comment ***************************************************************
* this macro stores *world* *cando* and *maxival on disk               *
***************************************************************)
(cond (*store(create-file iofile)
(insert-entries iofile '(*world* *cando* *maxival *randcount *last]



(defun drawstats ()
(comment ***************************************************************
* this macro draws the basic labels for the screen statistics         *
***************************************************************)
(gr_text 150 40 "function:   " 0 1 1 0 0 0 255)
(gr_text 150 20 "you are at:" 0 1 1 0 0 0 255)
(gr_box 148 0 155 60 0 0 255)
)



(defun typefunc (funct)
(comment ***************************************************************
* this macro populates the statistics box                             *
***************************************************************)
(gr_rect 230 2 72 56 255 255 255)
(gr_text 230 40 funct 0 1 1 0 0 0 255 )
(gr_text 230 20 *last 0 1 1 0 0 0 255 )
(gr_flush))
(comment *print options*) (help)
```