Models, algorithms and data structures for associative information systems

Mukerjee, Prithwis, Ph.D.

The University of Texas at Dallas, 1989

Models , Algorithms and Data Structures for Associative Information Systems

by

Prithwis Mukerjee. B.Tech. (Hons), M.S.

DISSERTATION
Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY
IN
MANAGEMENT SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS
AUGUST 1989

MODELS, ALGORITHMS AND DATA STRUCTURES FOR ASSOCIATIVE INFORMATION SYSTEMS

Approved by the supervisory committee

[illegible]

Dr. R. Chandrasekaran, Chairman.

E.E. Anderson

Dr. Evan E. Anderson.

[illegible]

Dr. Yu-Min Chen.

[illegible]

Dr. Syming Hwang.

Dedicated

to my parents

Subhrendu & Reba Mukerjee

and

with all my love

to my wife

Indira

Acknowledgements

This dissertation owes a great deal to my adviser. Prof. Ramaswamy Chandrasekaran who helped me to identify this topic and was a constant source of inspiration and insight. Without his continuous support it would have been difficult for me to complete this work. Prof. Evan Anderson was another source of encouragement and his research seminars helped me to clarify my thoughts and iron out wrinkles from this work at various stages. Prof. Yu-Min Chen and Prof. Syming Hwang looked at and commented on my work and their suggestions have been very useful.

My fellow graduate students at the university offered numerous suggestions and I would like to take this oppurtunity to thank Mr. Kala Seal. Mr. Adelmiro D'Costa. Ms. V. Revathi Iyengar. Dr. Lakshmi Narasimhan. Ms. Lakshinarayan Sambhavi. Mr. V. Padmanabhan. Mr. Joyendu Bhaduri. Mr. Swaminathan Sridhar. Mr. Amit Mukhopadhyay and others who have helped in various ways.

A dissertation is not the product of just the last few years of graduate study. It is the result of various inputs I have received over a course of time in various schools and colleges both in this country and in India. Of the many people who have had a significant influence on me I would like to mention Mr. Rosario. Mr Subramanium .Mr. S.P.Bhanj Deo. Fr. Hous. s.j.. and Fr. Desbrulais. s.j. of St. Xavier's School Calcutta. Fr. L. Hincq. s.j. of St. Xavier's College. Calcutta. Prof. B.G.Ghosh. Prof. S. Majumder and Prof. Amalendu Mukherjee of IIT. Kharagpur and Prof. Shun-Chen Niu of UT-Dallas. To these teachers and to all my other teachers I bow my head in gratitude.

Models , Algorithms and Data Structures for Associative Information Systems

Publication No. ___________

Prithwis Mukerjee, B.Tech (Hons) , M.S. ,Ph.D.

Supervising Professor : Dr. R. Chandrasekaran.

This dissertation seeks to draw upon certain aspects of human memory to design a method to store and retrieve information in an efficient manner. Information is stored associatively, i.e., it is not stored at particular locations but in terms of sets or associations of smaller pieces. In the human memory , concepts are stored in terms of a set of underlying features. Building on this premise, this work presents algorithms and data structures to implement the storage of concepts on a digital machine. The time required to retrieve information is calculated and is shown to be quite small.

In the second part of the dissertation, another aspect of human associative memory is considered - the ability of one thought, or concept, leading to another. It is shown that the models described in the first part of the dissertation can be used to provide a scheme to manipulate information between short term memory and long term memory. This means that information which would be required in a short time can be , in a way, anticipated, and moved into short term memory so that its retrieval can be faster. A small prototype of this system has been designed and implemented in Pascal.

Table of Contents

List of Figures

List of Tables

List of Graphs

Chapter 1

Introduction

The information processing revolution that has occurred during the last few years has completely changed the face of cognitive psychology. There has been an increase in the use of concepts taken from computer science for modelling psychological processes. This application of the computational metaphor is most clearly described by Boden (1979) who states that this metaphor can help the psychologist generate and test hypotheses. Computer programming languages can be used as formal ("mathematical") languages for expressing theories of human mental processes (Simon, 1979). Computational models of cognition and perception have been explored further by McClelland and Rumelhart (1986).

This process, however, is a two way street. Well tested ideas from psychology can also be used to design computer systems and develop applications. Most of our current ideas about computing are derived from our experience with conventional digital computers. There is however a widespread feeling (Hinton and Anderson, 1981) that computers are not a good model of how cognitive processes are embodied in the brain. Tasks like arithmetic and flawless memory for a large number of unrelated items are easy for computers but very difficult for humans. Conversely, tasks like perceiving a three dimensional world or recalling items from partial descriptions come naturally to humans but are incredibly difficult to implement on a digital computer. It seems that the computational processes in the brain are very different in kind from those in today's computers. People in many different fields, puzzled by the question of how the brain computes, have been struggling to formulate models of computation that mimic the human thought process. Artificial intelligence, neural networks and associative memory are areas of such research. This research is not interested, per se, in understanding how the brain works. but is dedicated to to the creation of automatic machines. faster computers and other similar systems.

Our current research is motivated by similar considerations. In a nutshell. what is attempted here is to design and develop an information system which can store a very large amount of information and yet it should be possible to navigate in this vast pool in a way that only relevant information is accessible. It is imperative that any retrieval process must work in time independent of the number of units of information stored. Further. once a piece of information has been located within the system. it should be possible to retrieve all other associated pieces of information very easily. To achieve this goal. two basic requirements have been set :

(1) Related pieces of information must be connected in some meaningful way.

(2) Overlapping pieces cf information must necessarily be stored but with the absolute minimum of redundancy.

The concept of human associative memory is historically derived from Aristotle's so called "Classical Laws of Association ". In essence. these laws state that mental items such as ideas. perceptions. sensations and feelings are connected under the following conditions : (Narayanan. 1984).

(1) if they occur in close proximity. ('spatial contact');
(2) if they occur in close succession ('temporal contact');
(3) if they are similar;
(4) if they are dissimilar;

Contemporary views of human associative memory generally use conditions (1) and (2) for storing or encoding information and conditions (3) and (4) for recalling information.

One way to look at associative memory is to imagine a collection of nodes physically linked together in a clever network. In our research , this paradigm is explored and implemented to design an information system to store city maps for automatic vehicle guidance. This is described in detail in chapter 6.

Associative memory could also be described as a collection or assemblage of elements having data storage capabilities which are accessed simultaneously and in parallel on the basis of data content rather than any specific address or location. (Hanlon, 1966). In this case information is not stored anywhere in particular, but is stored everywhere. Particular neural units do not store pieces of information but information is stored in the relationships among the units and each unit participates in the storage of many many "memories". chapters 3, 4 and 5 describe an information system which incorporates this concept.

Chapter 2

Review of Literature

To go about the design of an information system which meets the criteria set out above, it is instructive to review the literature which discusses various models of the human memory. Despite years of research, much of the brain remains a mystery, but it is believed that structural and chemical changes must occur as the result of the acquisition of new knowledge. Somehow, cortical neurons alter their reaction patterns to the external events which the organism comes to recognize and remember. There is a reasonably good agreement that permanent storage of information takes place either through chemical and structural changes but the ongoing activities of thougnt, conscious processes and the immediate memories are mediated through electrical activity. (Lindsay and Norman , 1977).

The semantic net (Quillian, 1968) was developed as an explicit model of human associative memory. A semantic network consists of nodes, which represents objects or situations, and arcs or arrows which represent the relationships between the nodes and which connect the nodes. Storage consists of representing the item using the nodes and arcs while recall consists of identifying a node or relation in the network and following the arcs involved to retrieve the structure and content of the original sequence.

Morton's (1970) logogen was perhaps the earliest real model to use the parallel processing metaphor. This functional model of memory was unusual on several counts because it attempted to distinguish between brain processes on the basis of clearly identifiable functional criteria rather than any structural criteria. Functional criteria included the logical nature of the code in which information is processed, the kinds of information that can interact and the logical form of the processing operation.

Anderson and Bower (1974) proposed an architecture of human associative memory (HAM) that is consistent with the traditional theory of processing. HAM receives physical signals and stores them in buffers. Parsers then attempt to identify and recognize the symbols, produce an input compatible with the memory representation of items and store it in working memory. An item in working memory is then passed to main memory and an output for the executive control mechanism is produced.

SHORT (Gilmartin et.al. , 1976) is a computer program written in SNOBOL which represents a theory of how humans use short term memory and, to a lesser extent, how they use the long term memory together with the various sensory related buffers during common short term memory tasks.

Feldman (1981) has developed a model of memory using information processing concepts that include : (a) parallel operation of subunits, (b) generalized lateral inhibitions, (c) an active semantic net, (d) positive and negative feedback and (e) generalized matching and relaxation. The major point of departure from conventional models is the explicit adoption of a connectionist framework as opposed to assuming that symbolic information is transmitted along some general channel.

Thinking has been thought of as a form of information processing and models of thinking should fit in with the more general frameworks for understanding human information processing. A number of broad frameworks (Bower, 1975 ; Hunt, 1971, 1973 ; Atkinson and Shiffrin, 1968) have been proposed but the difference between such schemes are generally a matter of detail and certain assumptions are widely accepted. These common assumptions have been incorporated by Gilhooly (1982) into his Modal model of thinking. According to this model, individuals have a vast long term memory and a small capacity working memory. Thinking is viewed as the manipulation of symbols both within working memory and between long term memory and working memory, according to some well defined rules.

The perceptron, originally developed by Rosenblatt (1958, 1962) and later by Minsky and Papert (1969) was intensely studied in the early 1960's. The basic element in these devices was the threshold logic unit (TLU) a particular type of the McCulloch - Pitt (1943) neuron. The TLU had a number of inputs , say n , each associated with real valued weight that plays a role analogous to the synaptic strength of a neuron. A TLU divides the n-dimensional space of possible input vectors into two regions , separated by a hyperplane, one region being associated with output 1 and the other with output 0. The values of the weights determine the position and orientation of the hyperplane.

Human perception is an extremely complex activity involving multiple interacting representations at many levels and a simple perceptron is clearly an inadequate model. Nevertheless perceptrons pave the way for a new approach to the problem. (Hinton, 1981, Minsky, 1977) This considers how computation might be organized within a device consisting of many interconnected perceptron like devices.

Minsky (1980) has also proposed the K-Line theory of memory which remembers an idea by creating a K-Line for it. When activated later, the K-Line induces a partial mental state resembling the one that created it. A partial mental state is a subset of those mental agencies operating at the moment. This view leads to many ideas about the development , structure and psychology of memory and about how to implement a frame-like representation in a distributed process.

Schank (1980) has presented a theory of Memory Organization Packets which serve as both processors and organizers of information in memory. This enables effective categorization of experiences in episodic memory, which, inturn, enables better predictive understanding of new experiences. As a continuation of this work Schank (1982) developed a system of dynamic memory based on scripts and schemas. A dynamic memory is one that can change its own organization and also learn.

Kolodner (1983a) discusses another type of dynamic memory. As new unanticipated items are added to memory it is able to reorganize itself and integrate the new items in its structure. The reorganization process maintains the memory's structure and also builds up knowledge retrieval strategies needed to search the structure. An algorithm is presented for knowledge based memory reorganization which includes processes for directed generalization and generalization refinement. Conclusions are drawn about maintaining accessibility in a conceptual memory, organizing generalized knowledge with respect to specialized knowledge and expected retrieval failures due to changes over time on the memory's organization.

The algorithm described above is used by Kolodner (1983b) in a reconstructive process used to model very long term episodic memory. This involves the application of four types of reconstructuion strategies. A component-to-context instantiation strategy is used to direct search to appropriate conceptual categories in memory. Component and context-to-context strategies search within the chosen conceptual category. The executive search strategies guide search for concepts related to the one targeted for retrieval.

Zenner et. al. (1985) have described a document retrieval system that permits a reduction of time needed to retrieve a document. fulfilling a user's query. as well as the amount of core space required for the document description relations. This paper is an extension of Radecki's (1979) work. based on lambda level fuzzy sets. and solves some of the time and space problems encountered in the earlier work.

Most of the systems described so far are currently in the conceptual stage though a few experimental prototypes do exist. Linear text systems . like indexed files. have been the mainstay of traditional information systems. All these systems need good keywords to perform efficiently. but a diversity of interests lead to a diversity of keywords. This leads Blair and Maron (1985) to conclude that the fraction of relevant material returned by keyword retrieval systems is less than 20 %.

Another extremely popular and widely used class of systems are the database management systems several of which are commercially available. All existing DBMS are based on one of the three data models - heirarchical. network or relational. Of the three. the relational model is by far the most popular and its origin in theoretical proposals is traced back to Codd (1970). Good descriptions of database systems are found in Ullman (1980) and Kent (1983). These systems. however. are generically handicapped because of the fixed nature of relationships they support. Joins do allow more relationships to be built up out of already existing relationships but totally random relationships between arbitrary entities cannot be supported. DBMS are excellent when many instances of relatively few relationships need to be stored., but they are very inefficient when just one instance of many relationships need to be processed.

Efforts to overcome the limitations of conventional information systems have resulted in what is known as non-linear systems. One of the most well known of such systems is the class of hypertext systems surveyed by Conklin (1987). These hypertext systems consist of text fragments embedded in a directed graph with labelled edges and instructions allowing the user to traverse edges.

Shasha (1985, 1986) has proposed another non-linear system known as Net-Book which consists of a body of knowledge in the form of text fragments plus a query language to help the user access appropriate fragments. Using database theory, hypertext systems knowledge representation and a study of textual fragments, called fragment theory, the system develops a data model to support knowledge exploration.

Two significant developments in computer science in the past decade have been relational databases and logic programming. Parsaye (1983) discusses the language PROLOG and its relation to relational databases.

A similar marriage of convienience results in expert database systems (EDS). This is a combination of a knowledge based expert system and a database system. EDSs can be used for developing applications requiring knowledge directed processing of shared information. An increasing number of applications including CAD/CAM, office automation and military command and control need this capability. Smith (1986) predicts that by the 1990's EDS would become one of the most important application development tools.

Expert database systems are one attempt to inject knowledge into databases but the semantic gap between conventional database systems on one hand and knowledge base management systems on the other is still very large. This is because, in the terminology of PROLOG, database systems provide means of managing facts but rules are not supported. Linneman (1986) has presented a new tool, CONSTRUCTORSET, based on a database programming knowledge, which allows rules to be supported. These rules are recursive and set oriented. They also support datatypes defined for the representation of updateable rules, thus providing an integration of fact and rule management using relational technology.

Malone et. al (1986) have described an intelligent system to help people share and filter information communicated by computer based messaging systems. It exploits concepts like frames, production rules and inheritance networks but avoids the unsolved problems of natural language understanding by providing with users with a rich set of semi-structured templates.

Kohonen (1984; and et.al. 1981) has concentrated on the principles of memory and learning by which certain elementary 'intelligent ' functions are performed adaptively, without external control, solely on the basis of received signals. The systems underlying these principles must be physical - and this is a significant restriction. This means that the basic components cannot implement arbitrary arithmetic algorithms even though these algorithms can be coded on a simple computer. The signal transformation must be simple and changes in the system variables must be smooth, continuous functions of time. This is a clear departure from conventional artificial intelligence approaches which are totally dependent on digital computers and high level languages.

Lenk and Floyd (1988) have examined the probability that the retrieved information is useful to the user's query. They present a sequential, Bayesian, probabilistic indexing model that explicitly combines expert opinion with data about system performance. The expert opinion is encoded into probability statements and these statements are modified by the users' feedback and the relevance of the retrieved information to the original queries. The predictive probability that a datum in the information base is applicable to the current query is a logistic function of the expert opinion and the feedback. The feedback enters the computation through a measure of association between the current query-datum with previous relevant query-datum pairs.

Maron [1982] in an opinion paper on associative search techniques describes two different ways to improve retrieval performance, viz., (a) appending associative search techniques to more or less standard (conventional) document retrieval systems and (b) designing document retrieval systems based on more fundamental and appropriate principles, namely, probabilistic design principles. He argues that the latter approach is more likely to yield better results.

Banerji [1962] has defined a concept as a class of objects whose members can be distinguished by processing its properties where property is defined to mean a partition of the set of all objects into disjoint classes and the definition is of a recursive nature. A concept is described by a list structure and a one-to-one correspondence is established between the recursive definition of a concept and its description list structure.

Chapter 3

Concepts : A Basic Model

This section presents a scheme to store and retrieve information based on a model of certain aspects of the human memory. It has long been conjectured that human memory functions by association. Seiffert et.al. (1986) have investigated recent theories about the representation of thematic information and conclude that two episodes that share a theme are connected together through a thematic structure.

information is better thought of as 'evoked' than as 'found'. Rather than imagining that particular neural units encode particular pieces of information. it is believed that information is stored in the relationships among the units and each unit participates in the encoding of many. many memories. What is further stored is a set of connection strengths. that is. how much does each unit contribute to a particular piece of information. Different individuals may store the same information as relationships among different units. It may also be the case that strengths differ even among same relationships. This leads us directily into the theory of concepts.

Classical Concept Theory

In linguistics. semantic theorists have attempted to specify certain features that were believed to be required for an idea to be conceived (Bierwisch. 1970; Katz. 1972). For example. the object <bachelor> might contain the semantic components < unmarried >. <adult>. < male >. An object that lacked the < unmarried > component might be referred to as a < man > or if the < married > component was present then it might be called a < husband >. Thus <bachelor> is a concept with these three features.

Classical concept theory requires that the features that represent a concept are

(a) singly necessary and

(b) jointly sufficient

to define that concept (Smith and Medin, 1981, pg. 23). With this in view, we can use set theory as a powerful representational device to model this phenomenon.

The basic associative model

Let us assume that we have stored N concepts in the system, where the Kth concept $C^{(K)} = \{ x_1, x_2, \ldots, x_p \}$, that is the p features $x_1, x_2, \ldots, x_p$ define the concept $C^{(K)}$. If M be the total number of features then $M = |C|$ where $C = C^{(1)} \cup C^{(2)} \cup \ldots \cup C^{(N)}$.

Given a query, defined as an input set of features $\{ x_1, x_2, \ldots, x_q \}$, the system must respond with one of the three decisions : (a) the input set does uniquely match one particular concept, (b) the input set does not correspond to any stored concept, (c) the input set matches more than one concept and the system is unable to distinguish among these.

The Algorithm

$X = \{x_1, x_2, \ldots\ldots, x_q\}$ the set of q input features. $q \geq 1$

```
Step 1 :
    Pick $x_{(1)} \in X$
    $S_{(1)} := \{C \mid x_{(1)} \in C\}$
    $S := S_{(1)}$
    if $|S| \geq 2$ then goto Step 2
                 else if $|S| = 1$ then stop with decision A
                                   else stop with decision B

Step K : $K \geq 2$
    $X := X - x_{(k-1)}$
    if $|X| = 0$ then stop with decision C
                 else pick $x_{(k)} \in X$
                      $S_{(k)} := \{C \mid x_{(k)} \in C\}$
                      $S := S \cap S_{(k)}$
                      if $|S| \geq 2$ then goto step K+1
                                   else if $|S| = 1$ then stop with decision A
                                                     else stop with decision B
```

where :

decision A implies that the system has uniquely identified a concept which matches the input pattern.

decision B implies that the system has determined that no stored concept matches the input pattern.

decision C implies that the system has run out of features to match and it cannot distinguish among the concepts still lying in set S.

Time Complexity

Let K_A, K_B and K_C be the number of steps required to stop with decisions A, B and C respectively. If K^* is the actual number of steps required then $K^* = \min(K_A, K_B, K_C)$.

$K_C = q$ because the number of steps required before the algorithm runs out of features to match and so arrives at decision C is equal to the number of input features.

Also $K_B > K_A$ because the cardinality of S never increases and decision A occurs when the cardinality is 1 and decision B occurs when cardinality is 0. So if at all it occurs, decision A occurs before decision B. Hence K_B is always greater than K_A.

Thus $K^* \leq \min(K_B, K_C)$

$\Rightarrow \quad K^* \leq \min(K_B, q)$

If q is very small, q forms an upper bound on K^*. However if q, the size of the input set, is very large then K_B forms the upper bound on K^*. So K_B forms a true upper bound on K^*.

Let	N	=	number of concepts stored in the system.
	M	=	number of features used to define the concepts.
	p	=	average number of features per concept
			.. .. a measure of the average size of each cocept.
	n	=	average number of concepts which share an idea
			.. .. a measure of how inter-related the concepts are.

then $x = n/N = p/M$ is a measure of *feature density* of each concept

and $x \ll 1$.

If we ignore the trivial case of a B decision in the first step then it has been shown in the appendix that

$E[K_B]$ = expected value of $K_B = 2 + n.x$

Hence $K^* = O(n.x) = O(n)$ since $x \ll 1$.

Each step of the algorithm, after step 1, involves the finding of the intersection of two sets, each with a maximal cardinality of n. With the sets stored as sorted linked lists each intersection takes $O(n)$ time.

So the overall, average case complexity of the algorithm is $O(n^2)$

Proof : $E[K_B] = 2 + nx$

The process is modelled as follows :

There is an urn containing N balls, numbered 1 through N. At each step a group of n balls is pulled out, the numbers are noted, and then the balls are put back again. The process is said to stop after K steps when there exists a group of T balls in the Kth draw which were present in all previous K-1 draws and all of which failed to appear in the K+1 st draw. T is random and can vary from 1 to n.

Evidently $K_B = K + 1$. So for the case under consideration K_B is greater than 1 and hence K is greater than 0. We are interested in $P[K = k]$.

Let $A_T(m)$ be the event that in the mth draw there were exactly T balls which were present in all previous m - 1 draws. Let $B_T(m)$ be the event that in the mth draw a particular group of T balls failed to appear.

$$\begin{aligned}
P[K=k] &= P\left[A_T(k) . B_T(k+1)\right] \\
&= \sum_{t=1}^{n} P\left[A_T(k) . B_T(k+1) \;/\; T = t\right] . P[T=t] \\
&= \sum_{t=1}^{n} \frac{P\left[A_T(k) . B_T(k+1) . T=t\right]}{P[T=t]} . P[T=t] \\
&= \sum_{t=1}^{n} P\left[A_T(k) . T=t\right] . P\left[B_T(k+1) . T=t\right]
\end{aligned}$$

The two events are independent because once a group of $T = t$ balls have been identified through event $A_T(K)$ at the K^{th} draw the probability of their appearance in the $k+1^{st}$ draw does not depend on whether they appeared in the previous draws.

$P[A_T(k) . T = t] = P[R_k = t]$ where R_k is the number of balls on the k^{th} draw which were present in all previous $k - 1$ draws.

$$\text{Let } x_i \begin{cases} = 1 & \text{if the } i^{th} \text{ ball in the } k^{th} \text{ draw has appeared all previous } k-1 \text{ times} \\ = 0 & \text{otherwise.} \end{cases}$$

$$R_k = \sum_{i=1}^{n} x_i$$

$\left(\frac{n}{N}\right)$ is the probability of a ball appearing in any draw.

So $P[x_i = 1] = \left(\frac{n}{N}\right)^{k-1}$ and x_i is a Bernoulli random variable.

Since the x_i 's are independent. R_k is a Binomial random variable with

parameters $n . \left(\frac{n}{N}\right)^{k-1}$

So $P[A_T(k) . T = t] = P[R_k = t]$

$$= \binom{n}{t} \cdot \left[\left(\frac{n}{N}\right)^{k-1}\right]^{t} \cdot \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n-t}$$

$P[B_T(k+1) . T = t]$ = the probability that the particular group of $T = t$ balls which were identified in the A_T event failed to appear in the $k + 1^{st}$ draw.

$$= \left(1 - \frac{n}{N}\right)^{t}$$

Hence P I K = k I

$$= \sum_{t=1}^{n} P\left[A_T(k) . T = t\right] . P\left[B_T(k+1) . T = t\right]$$

$$= \sum_{t=1}^{n} \binom{n}{t} . \left[\left(\frac{n}{N}\right)^{k-1}\right]^{t} . \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n-t} . \left(1 - \frac{n}{N}\right)^{t}$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \sum_{t=1}^{n} \binom{n}{t} . \left[\frac{\left(\frac{n}{N}\right)^{k-1} . \left(1 - \frac{n}{N}\right)}{1 - \left(\frac{n}{N}\right)^{k-1}}\right]^{t}$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \sum_{t=1}^{n} \binom{n}{t} . Q^{t} \qquad \text{where } Q = [\ldots]$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \left[(1+Q)^{n} - 1\right]$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \left[\left\{\frac{1 - \left(\frac{n}{N}\right)^{k-1} + \left(\frac{n}{N}\right)^{k-1} . \left(1 - \frac{n}{N}\right)}{1 - \left(\frac{n}{N}\right)^{k-1}}\right\}^{n} - 1\right]$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \left[\left\{\frac{1 - \left(\frac{n}{N}\right)^{k-1} . \left(1 - 1 + \frac{n}{N}\right)}{1 - \left(\frac{n}{N}\right)^{k-1}}\right\}^{n} - 1\right]$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n} . \left[\left\{\frac{1 - \left(\frac{n}{N}\right)^{k}}{1 - \left(\frac{n}{N}\right)^{k-1}}\right\}^{n} - 1\right]$$

$$= \left[1 - \left(\frac{n}{N}\right)^{k}\right]^{n} - \left[1 - \left(\frac{n}{N}\right)^{k-1}\right]^{n}$$

Since $x = \left(\frac{n}{N}\right)$

So $P[K = k] = \left(1 - x^{k}\right)^{n} - \left(1 - x^{k-1}\right)^{n}$

$$
\begin{aligned}
P[K \le s] &= \sum_{k=1}^{s} P[K=k] \\
&= (1-x^1)^n - (1-x^0)^n \\
&\quad + (1-x^2)^n - (1-x^1)^n \\
&\quad + (1-x^3)^n - (1-x^2)^n \\
&\quad \cdot \\
&\quad \cdot \\
&\quad + (1-x^s)^n - (1-x^{s-1})^n \\
&= (1-x^s)^n - (1-x^0)^n \\
&= (1-x^s)^n
\end{aligned}
$$

$$\lim_{s \to \infty} P[K \le s] = \lim_{s \to \infty} (1-x^s)^n = 1 \quad \text{because} \quad x \ll 1$$

$$E[K] = \sum_{k=1}^{\infty} k.P[K=k] = \lim_{s \to \infty} \sum_{k=1}^{s} k.P[K=k]$$

$$
\begin{aligned}
\sum_{k=1}^{s} k.P[K=k] &= 1.(1-x^1)^n - 1.(1-x^0)^n \\
&\quad + 2.(1-x^2)^n - 2.(1-x^1)^n \\
&\quad + 3.(1-x^3)^n - 3.(1-x^2)^n \\
&\quad \cdot \\
&\quad \cdot \\
&\quad + s.(1-x^s)^n - s.(1-x^{s-1})^n \\
&= s(1-x^s)^n - \left[(1-x^{s-1})^n + (1-x^{s-2})^n + \dots + (1-x^1)^n + (1-x^0)^n\right] \\
&= s(1-x^s)^n - \sum_{t=1}^{s-1} (1-x^t)^n \\
&= s(1-x^s)^n - \sum_{t=1}^{s} (1-x^t)^n + (1-x^s)^n \\
&= (s+1).(1-x^s)^n - \sum_{t=1}^{s} (1-x^t)^n
\end{aligned}
$$

$$= \quad (s+1).(1-x^s)^n \quad - \quad \sum_{t=1}^{s}(1-nx^t) \qquad \text{because } x \ll 1$$

$$= \quad (s+1).(1-x^s)^n \quad - \quad s \quad + n.\sum_{t=1}^{s}x^t$$

$$= \quad (s+1).(1-x^s)^n \quad - \quad s \quad + n.x.\frac{1-x^s}{1-x}$$

Hence $E[K] = \lim_{s\to\infty} (s+1).(1-x^s)^n - s + n.x.\frac{1-x^s}{1-x}$

$$= \quad s + 1 - s + \frac{nx}{1-x} \qquad \text{because } x \ll 1$$

$$= \quad 1 + nx$$

Since $K_B = K + 1$ therefore $E[K_B] = 2 + nx$

Implementation

A small version of the model, coded in Waterloo pascal has been implemented on an IBM 4381 computer. M = 100 features were defined and numbered 1 through 100. Random combinations of 10 features were defined as concepts and numbered consecutively. An array of size 100 was used to represent the features and a linked list running out of each element of the feature array stored the concepts, of which this feature was a part, in a sorted manner. Additionally, a circular linked list, linked together the features which corresponded to a particular concept [Fig. 1]. The system worked as expected, though the size of the sample was too small to corroborate meaningfully, the probabilistic analysis.

Experiments were conducted to get an idea of the storage space required to store the data in the manner described above and also to determine the order of time required to set up the database. This was done by causing the system to go through a 'dry run' , that is, the data was just stored but no attempt was made to retrieve it. Two sizes of concepts were considered and there were three data sets for each size of concept. This gave six observations of time and storage space used. The actual values were obtained from the listing file generated by the pascal compiler and are given in Table 1. Plots are shown in Graphs 1 and 2.

Legend :

squares represent features
dark polygons represent concepts

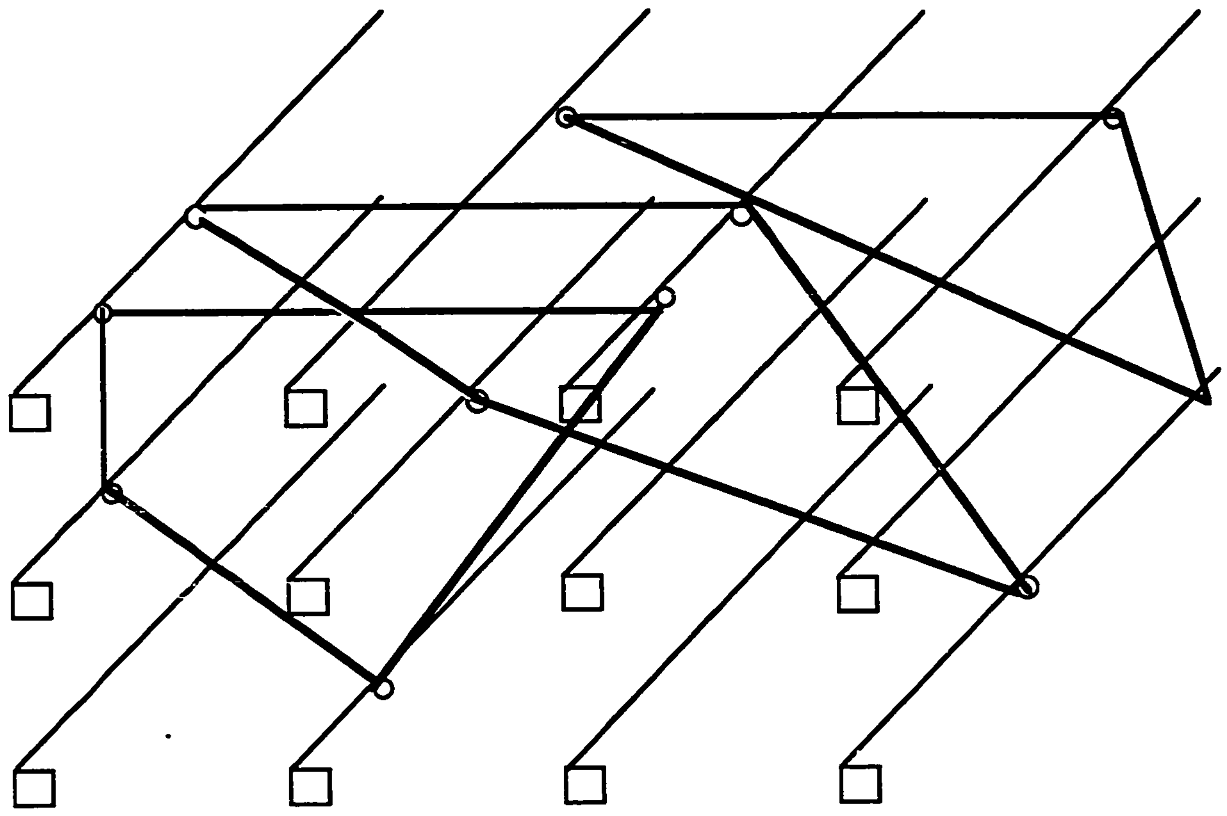


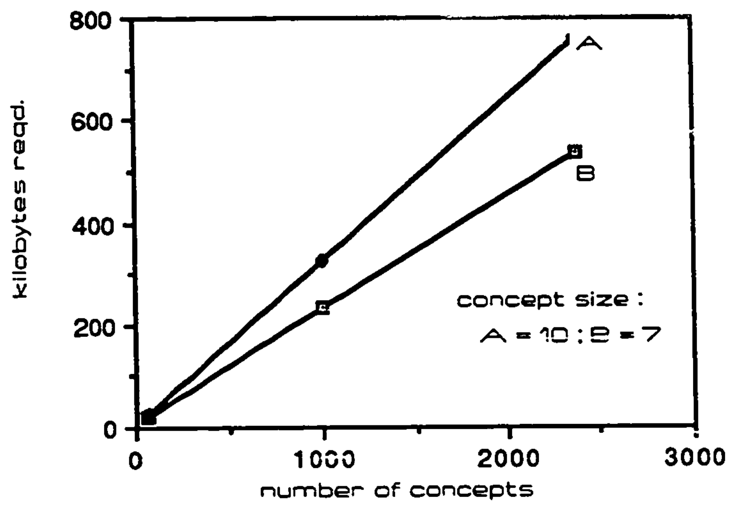
Figure 1 : Data Structure for Model

		number of concepts stored		
		50	1000	2368
size of concept	7	0.9 21.88	126.15 232.51	673.01 535.78
	10	1.47 26.68	243.49 327.42	1313.9 760.19

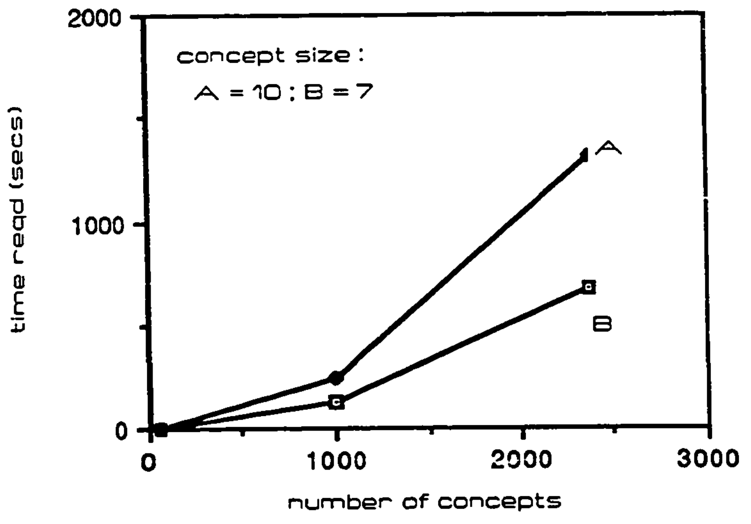
note : in eaach box, the upper value represents time in seconds and the lower represents memory requested in kilobytes.

program run on IBM 4381 running VM/SP rel.4

Table 1 : Time and memory requirements for storing data.



Graph 1 : Memory required to store data



Graph 2 : Time required to store data

Properties of the Model.

This model and the associated algorithm has certain remarkable properties.

(a) The retrieval time is independent of N, the total number of concepts stored in the system, as well a M, the number of features used to define the concepts. In general both of these would be very large numbers.

(b) The retrieval time depends on n , the average number of concepts which share a feature - a measure of how interrelated the concepts are. Intuitively, this makes sense because it is quite difficult to distinguish between identical twin children precisely because they share a large number of features. Direct experimental verification of this intuition is difficult. However Srull et.al (1985) in an article describing a general associative storage and retrieval theory of human memory claim that incongruent events are best recalled. This can be said to be a partial validation of the current model.

One source of error is the possibility that the cardinality of the set S reduces to 1, the system concludes that it has uniquely identified the concept and stops, but the remaining features, which have not been explored, are not contained in the concept so identified. To guard against this, the system could be made to go through r more steps. If the concept was indeed identified correctly, the cardinality would still remain 1. Otherwise two things can occur (a) in one of these steps, the cardinality would fall to 0 and the system would correctly conclude that no concept matches, or (b) by a freak chance the next r features might still be contained in the wrongly identified concept. The probability of this happening is $(p/M)^r$ and since $p/M = x \ll 1$, a small positive value of r would serve to reduce the probability of this error below any specified significance level. In fact this is consistent with the view that even humans are sometimes confused and arrive at erroneous conclusions.

Another interesting feature which the current implementation of the system exhibits is the human ability to let one's thoughts 'ramble'. By following the cyclic linked lists, one can visit all the features that a concept contains and so all the other concepts which share these features. Concepts which overlap to a large degree are deemed to be 'closer' to the input concept and may be explored profitably. This is explored in greater detail in chapter 5.

Chapter 4

Concepts : A Generalized Model

The classical concept theory may run into problems both from a psychological as well as from an information processing point of view. The classical theory requires that the features be singly necessary but this need not be true. Some features may be salient while others may be insignificant. When translated into information processing terms, this means that the basic model is rigid and inflexible.. An input candidate would not be identified as an existing concept even if one insignificant feature is absent. This is not desirable because the system should be able to identify input candidates even if there is a match of significant features.

General Concept Theory

In this version of concept theory, the features that represent a concept are salient ones that have a substantial probability of occuring in instances of the concept. More precisely, if X is a feature of C then (Smith and Medin , 1981, pg. 62) :

(a) X is salient (either perceptually or conceptually)

(b) conditional probability $P (X / C)$ is high.

Since features can vary in both their salience and their probability, we explicitly need to indicate these variations in the representation. So each feature of the concept is accompanied by a weight that reflects the combined salience and the conditional probability.

The standard extensional model of the fuzzy concept theory is the original fuzzy set model proposed by Zadeh (1965) and later formalized by Osherman and Smith (1981) and Zadeh (1982). For our information processing model a simplified version of this theory would be used.

The Generalized Associative Model

In this model the concept is defined as $C_j = \{\langle x_1, u_j(x_1)\rangle, \langle x_2, u_j(x_2)\rangle, \ldots, \langle x_p, u_j(x_p)\rangle\}$ where $u_j(x_i)$ represents the weight of feature i in concept j. In a computable model of the type attempted here, it is imperative to give a mathematical interpretation to the weight. Even though an axiom of present day fuzzy set theory (Zadeh, 1978) insists that fuzziness is not the same as randomness, Hisdal (1988) has argued that probabilities can indeed be used to develop the weights, or grades of membership as they are called in fuzzy set theory. In view of this and in view of the difficulty of quantifying salience we define a concept as follows : $C_j = \{\langle x_i, u_j(x_i)\rangle \mid u_j(x_i) = P(x_i / C_j) \geq \varepsilon_0\}$

Since we do not require either $\sum_i P(x_i / C_j)$ or $\sum_j P(x_i / C_j)$ to add up to 1 we shall avoid the use of the term probability and instead use *possibility*.

Given a query, defined as an input set of features ($x_1, x_2, \ldots, x_q$), the system must respond with one of the three decisions : (a) the input set does uniquely match one particular concept, (b) the input set does not correspond to any stored concept, (c) the input set matches more than one concept and the system is unable to distinguish among these.

The Algorithm

```
$X = \{x_1, x_2 \ldots\ldots\ldots x_q\}$ the set of q input features
$S^{(0)} = \emptyset$
Step i :
  Pick $x_i \in X$
  $S^{(i)} := \left\{ \left\langle C_j, \vee(C_j)^{(i)} \right\rangle \middle| \begin{array}{l} \left[\left(\left\langle C_j, \vee(C_j)^{(i-1)} \right\rangle \in S^{(i-1)}\right) \vee \left(\left\langle x_i, u_j(x_i)\right\rangle \in C_j\right)\right] \\ \wedge \left[\vee(C_j)^{(i)} = P(C_j / x_i)\right] \wedge \left[\vee(C_j)^{(i)} \geq \varepsilon_1\right] \end{array} \right\}$

      where $P(C_j / x_i) = \dfrac{P(x_i / C_j) \cdot P(C_j)}{\sum P(x_i / C_k) \cdot P(C_k)}$
            $\forall k \ni [x_i, u_k(x_i) \in C_k] \vee [C_k, \vee(C_k)^{(i-1)} \in S^{(i-1)}]$

      and   $P(x_i / C_k) = u_k(x)$   if $\langle x_i, u_k(x_i)\rangle \in C_k$
                          $= \varepsilon_0$   otherwise

      and   $P(C_j) = \vee(C_j)^{(i-1)}$   if $\left\langle C_j, \vee(C_j)^{(i-1)} \right\rangle \in S^{(i-1)}$
                    $= \varepsilon_1$   otherwise

  if $\exists j \ni \left[\left\langle C_j, \vee(C_j)^{(i)} \right\rangle \in S^{(i)}\right] \wedge \left[\vee(C_j)^{(i)} \geq \omega\right]$
      then stop with decision A
      else $X := X - x_i$
           if $X = \emptyset$ then if $S^{(i)} = \emptyset$ then stop with decision B
                                                   else stop with decision C
                            else goto step i + 1
```

where :

decision A implies that the system has uniquely identified a concept which matches the input pattern.

decision B implies that the system has determined that no stored concept matches the input pattern.

decision C implies that the system has run out of features to match and it cannot distinguish among the concepts still lying in set S.

What this means is that at each step we consider the concepts from the previous step, together with the concepts proposed by the current feature and calculate the possibility of their match with the input data, conditioned on the presence of the current feature. If any such possibility falls below a certain threshold value, the corresponding concept is temporarily removed from consideration.

As more and more features of the input set match with the features of a particular stored concept, the possibility that the input set matches with the stored concept becomes higher and higher. Alternatively if a feature of the input set is not found in a stored concept some features of which had matched in the earlier steps, the possibility of a match is lessened. If a large number of highly significant features, that is those with high u values, have been matched earlier, the possibility of a concept match is high and the lack of few unmatched features would not lessen it significantly. Also if a few insignificant features have matched earlier and this is followed by a series of non-matches, the possibility would fall below ε_1 and the concept would be temporarily removed from consideration.

In this model the cardinality of the set S would decrease much more slowly and we could expect more decisions of type C. However instead of just saying that the system is incapable of choosing among the concepts still present in S, the system would now have a ranking on the concepts based on the possibility values, with the one having the maximum possibilty being the most likely match.

In the basic model the order in which the features of the input set are considered is immaterial. In this case however, the order is indeed important because a salient and highly probable feature may not be considered initially and the system might stop at an erroneous A decision before the feature is considered.

To guard against this, (q−i+1) versions of $S^{(i)}$ are generated at each step i, one corresponding to each remaining feature. Each of these sets $S^{(i)}_t$, t = 1, 2, ..., (q−i+1) are compared with $S^{(i-1)}$ to determine the total absolute change in possibilities as shown below

$$\Delta_t = \sum_k \text{abs}\left[\max\left\{\nu(C_k)^{(i-1)}, \varepsilon_1\right\} - \max\left\{\nu(C_k)^{(i)}, \varepsilon_1\right\}\right]$$
$$\forall k \ni \left[\left\langle C_k, \nu(C_k)^{(i-1)}\right\rangle \in S^{(i-1)}\right] \vee \left[\left\langle C_k, \nu(C_k)^{(i)}\right\rangle \in S^{(i)}_t\right]$$

The $S^{(i)}_t$ having the maximum Δ_t is chosen as the $S^{(i)}$. This ensures that the important features, that is the ones having the maximum impact on the possibilities, are considered first.

Time Complexity

In the worst case, the number of steps required is equal to the number of features in the input, that is, q.

At each step the maximum size of $S^{(i)} = \frac{1}{\varepsilon_1} = n_0$

So in each step, the construction of $S^{(i)}_t$ needs to look at n_0 concepts from $S^{(i-1)}$ and n concepts with which the feature x_i is associated, to give a maximum of $(n_0 + n)$ unit operations. However, (q−i+1) sets of type $S^{(i)}_t$ need to be generated at each step i, so that the total number of operations in each step is $(q-i+1)(n_0+n)$ So the total number of operations is

$$\sum_{i=1}^{q} (q-i+1)(n_0+n)$$
$$= q(q+1)(n_0+n) - \frac{q(q+1)}{2}(n_0+n)$$
$$= (q+1)(n_0+n)\frac{q}{2}$$
$$= O(q^2 n_0 + q^2 n)$$

Lower Bound

We note that there is a lower bound on the number of steps before the algorithm stops with one of three possible answers. Whatever clever method is used to choose the order in which the features are considered. one can , with suitable data , force the system to go through at least q steps. To see how this is true. consider the possibility that there exist two concepts which share q or more features but this particular set of features is not found in any other concept. Now if q of these features constitute the input set. then no system can distinguish between these two concepts in less than q steps and then too it can only arrive at a C decision.

We have already noted that each step we need look at $(n_0 + n)$ concepts to generate set $S^{(i)}$. Thus the lower bound is given by $O\ (qn_0 + qn)$

Assigning Possibilities.

The generalized model described so far leaves us with the problem of assigning possibilites to each feature in a concept. As asserted earlier. the possibility of a feature in a 'concept' is somewhat analogous to the grade of membership of an item in a fuzzy set. The concept of grade is bound up with the general problems of fuzzy reasoning in fuzzy set theory.

Giles (1988) approaches this problem by treating assertions rather than sentences as fundamental. where an assertion is characterized in terms of utiltiy and decision theory. Dubois and Prade (1980) have surveyed a number of guidelines on developing membership functions for fuzzy sets. The sets based on statistics are perhaps some of the most naturally fuzzy sets that can be used. Civanlar and Trussel (1986) present guidelines to construct membership functions for fuzzy sets whose elements have a defining feature with a known probability density function defined on the universe of discourse.

Hisdal (1988) has proposed the 'TEE model' for grades of membership. Instead of starting out from mathematical postulates such as assumed min and max operators for the AND and OR functions. this model uses a semantic. physio-logical. psycho-logical starting point by investigating the possibilities which a person has for assigning linguistic labels and partial grades of membership in a meaningful way.

Even if there was at one's disposal a methodology to assign possibilities. and that is far from true. the possibilities associated with each feature in a concept would remain constant. This is an unfortunate restriction because the importance of a feature in a concept could very well differ with individuals or organizations and it would be impossible to assign one set of weights without seriously affecting the efficiency of the system. As an example. consider the feature *unionized* which could be part of either a concept involving labour unions or one involving ionized gases. A sociologist would expect the former concept to be matched whereas a physicist would expect the latter.

To avoid both these problems we propose an adaptive system which learns with usage. To begin with it is assumed that a method is available which assigns possibility values. This method need not be theoretically sound. nor is it expected to generate very accurate values. However the possibility values are allowed to change with usage. Then if two identical systems are installed. one in the sociology department and another in the physics department. the two systems should get customized to the two environments. This is done as follows :

Suppose an input sequence $X = \{x_1. x_2 \ldots\ldots x_q\}$ causes the system to stop. in the worst case. after q steps with a C-type decision. and the set $S^{(q)}$ contains j concepts $C_{(1)}. C_{(2)} \ldots\ldots C_{(j)}$ ranked according to their possibilities. However the user indicates that his query matches concept i . where i is between 2 and j. Let X_0 be the set of features which were used to identify $C_{(i)}$. that is

$$x_0 = \left\{ x \,\middle|\, [x \in X] \wedge \left[\exists\, u_i(x) \ni \left(x. u_i(x)\right) \in C_{(i)}\right]\right\}$$

Then the following possibilities are changed as follows

$$u_k(x) = (1+\alpha)u_k(x) \quad \forall k,x \ni [x \in X_0] \wedge [(x \,.\, u_k(x)) \in C_{(i)}]$$
$$u_k(x) = (1-\alpha)u_k(x) \quad \forall k,x \ni [x \in X_0] \wedge [(x \,.\, u_k(x)) \notin C_{(i)}]$$

This would ensure that the next time an input set X contains $X_0 . C_{(i)}$ would get a higher ranking. Thus with usage, the system would adapt itself to the needs of the user by building up the correct possibility values.

Chapter 5

Associative Retrieval

The mind proceeds along its path leaving traces of its progress. When the neural circuits responsible for the thought processes operate they are said to become active. Even though the neural processes of thought and memory are not yet completely understood, it is evident that once a sufficient amount of activity has been started , it is difficult to terminate it. As an example, it is often quoted that the command "Do NOT think about a pink elephant" has precisely the opposite effect on the subject.

Whatever the mechanism, whenever the thought and memory processes become active, they tend to keep going by themselves. Once a memory structure has been activated, it remains more accessible for future use. The attempt to retrieve one detail from memory invariably brings out a host of other details whether they are wanted or not.

This subconscious activity ties up normal processing resources. Further this activity is of an arbitrary or random nature. The subconscious mechanism is definitely not as powerful as the conscious activity of directed thought. The subconscious mechanism works away following paths through memory structures, activating arbitrary nodes and setting off new pathways. They do not seem capable of intelligent assesment of what they have done nor of intelligent decision at crucial points. (Lindsay and Norman, 1977). This random rambling is not without benefits. Associative theories of thought depend on this to explain many phenomena.

Mednick (1962) defines creative thinking process as the forming of associative elements into new combinations which are in some way useful and suggests three ways in which ideas could be brought together : (a) serendipidity - where the accidental occurence of certain stimuli evoke the required concepts in close proximity. (b) similarity - where the required elements may be evoked together by similarity on some dimension and (c) mediation - where the required concepts are evoked through the mediation of certain associates they have in common. The last method is particularly interesting because it could bring forth creative ideas like using a vacuum cleaner to remove a cloud of flies from that have settled on the ceiling. Concepts like "ceiling - floor" and "floor - vacuum" are used to generate "ceiling - floor - vacuum" concept and then the "ceiling - vacuum" concept. "Floor" was a feature that mediated the link.

Koestler (1964), has put forward a somewhat more sophisticated associative theory. This bisociative theory claims that a creative act involves linking together two previously unconnected "frames of reference" (also known as "associative contexts", "universes of discourse" or "matrices"). The case of Archimedes solving the problem of measuring the volume of an irregular object serves to illustrate the bisociation of different associative contexts. Koestler has interpreted other examples of scientific creativity, for example, discovery of vaccination, notions of elliptical planetary orbits and evolution by natural selection, as cases of bisociation but no predictions that could be tested or computable models have been proposed.

Departing the lofty heights of creative thought, we address our information processing problems. The long term memory capacity of human beings is prodigal. Landauer (1986) has tried to estimate the functional information content of human memory. Using a method that depends on measured rates of input and loss from long term memory and on an analysis of the informational demands of memory based performance. it is estimated that approximately 10^9 pieces are stored.

Evidently. any practical information system cannot function effectively with a random access to such a huge memory. The information overload would cripple it. According to the modal model of Gilhooly (1982). discussed earlier. individuals have a vast long term memory and a small capacity short term memory. Thinking is seen as the manipulation of symbols both within working memory and between long term and working memory. When a person is solving an algebra problem. information regarding mathematical manipulations is 'at the top of the head'. But again when the same person is driving a car down a busy freeway. information about driving replaces earlier information at the 'top'. So any successful information system should be able to function as a filter. or a seive. to pull out only relevant concepts from the vast store of memory.

There are two standard types of errors that plague most conventional document retrieval systems [Maron. 1982]. which need to be taken care of in any system which we design. These could be called "errors of precision" and "errors of recall". The first kind of retrieval error is where an irrelevant document is retrieved and in the second a relevant document is not retreived. If the value of a relevant retrieval is much greater than the costs involved in retreiving an irrelevant document then the second type of error is more serious. Kuhn and Maron [1960] gave two ways of dealing with this second type of error : (1) by defining a measure of similarity-of-meaning between terms in a document it is possible to widen a search query in order to capture relevant documents that might not otherwise be retrieved. (2) by defining a measure of similarity-of-content between documents it is possible to widen the search query as well to achieve the same purpose.

Information Filtering

We shall allow the system to ramble from concept to concept. much like the subconscious activity described in Lindsay and Norman (1977). To do so we shall use Mednick's suggestion of mediation by using features as bridges between concepts. Finally we shall use Koestler's bisociative theory and Mendick's original definition of creativity to generate new concepts.

What is interesting is in its 'rambles' through a sea of concepts. the system will fish out a bunch of associated concepts which can now be put 'at the top of the head' or. in more formal terms. in a short term memory. This is very similar to the second method given by Kuhn and Maron to widen the search. To draw a parallel. what is a document in their research is referred to as a concept here and the ideas in this work correspond to the terms that define their document.

Concepts overlap as they share features. This can be viewed as a graph where the concepts are nodes and arcs connect the nodes or concepts which overlap. Associated with each arc is a strength which indicates the degree of overlap. that is the number of features shared between the two. Such a graph would be undirected and could have a very large number of arcs because it is possible for a concept to share a one or two features with many other concepts. We however are interested in arcs which represent a strong overlap. that is only those which strongly associated concepts. Besides being intuitively reasonable this makes the scheme more amenable to computation. So each undirected arc is replaced by two directed arcs of the same strength but running in opposite directions. Next the strengths of all the arcs directed out of a node are checked to determine the maximum. and only those arcs whose strength is equal to this maximum are retained while the rest are discarded.. When done for all the nodes. this results in a directed graph with relatively fewer arcs.

Note that an arc pointing from A to B does not necessarily imply an arc pointing from B to A. This is because, to take an extreme example, A could be an unusual concept which shares just one feature with B and nothing with anybody else. Hence there would be an arc from A to B. B however could have many more neighbours with higher degrees of overlap and thus the arc from B to A would be discarded. But if A and B share a large number of features then it is quite possible that there would be arcs running in both directions.

We now have a structure in which we have concepts and arcs which point to their most strongly associated neighbours. If we follow the arcs we are in effect following a 'train of thought' which is similar to the subconscious activity of Lindsay and Norman (1977).

Once we have a directed graph structure various search schemes could be used to traverse it and generate ' trains of thought '. For reasons explained in the next section, a straight forward depth first search (dfs) scheme proved inadequate and a modified dfs, designated equi-balance search, is used. For the moment we assume that an equi-balance search (ebs) begins from a concept node, traverses the graph to a certain extent and stops. In the course of the traversal as each node is visited, the features of the corresponding concept are identified. A weight is attached to each such feature and these are put into a buffer. If such a feature was already present in the buffer, the cumulative weight of the particular feature in the buffer increases. Thus at the end of the traversal we have a buffer consisting of a set of features together with their respective cumulative weights.

Weights decrease as the system delves deeper into the search tree. This is because the depth is a rough measure of how 'distant' or dissimilar the current concept is from the root or starting concept and the features it contains should have a lesser weight.

We next describe a major cycle in our filtering scheme. Input to a major cycle consists of a 'pole' or starting concept and the cycle consists of the following steps :

(1) flush out the feature buffer so that it is empty

(2) beginning with the pole concept perform an equi-balance search

(3) in the resulting feature buffer, rank the features according to their cumulative weight and choose the p most heavily weighted features, where p is the average number of features in a concept.

(4) the p features so chosen could correspond to a concept which already exists, but if they do not, define a new concept with these features. In any case the resulting concept becomes the pole concept for the next major cycle.

Incidentally the emergence of a new concept ties in neatly with Mednick's definition of creativity.

The main filtering scheme can now be described in terms of major cycles. The process is initiated when the user inputs a set of features and requests a concept to be identified. The retrieval scheme described in chapters 3 and 4 are used to do so. If a successful identification is possible this becomes the first pole concept. Otherwise the set of input features is defined to be the first pole concept. In any case a sequence of major cycles is now initiated. The ebs traversals embedded in each major cycle 'fillter out' the concepts which are visited and these are pushed into a working or short-term memory. The process stops when any of the following three events happen : (1) Two successive pole concepts are identical. In this case the short-term memory would not get any more new concepts. (2) The number of major cycles reach an upper limit set by the storage capacity of the short-term memory. (3) The system is interrupted by a fresh set of features given by the user.

This filtering scheme is appealing because it is in agreement with Gilhooly's (1982) modal model which deals with the manipulation of symbols between long-term and short-term memory. It anticipates future information requests by pulling out information relevant to the current request. Subsequent queries can then be serviced more easily.

Equibalance Search

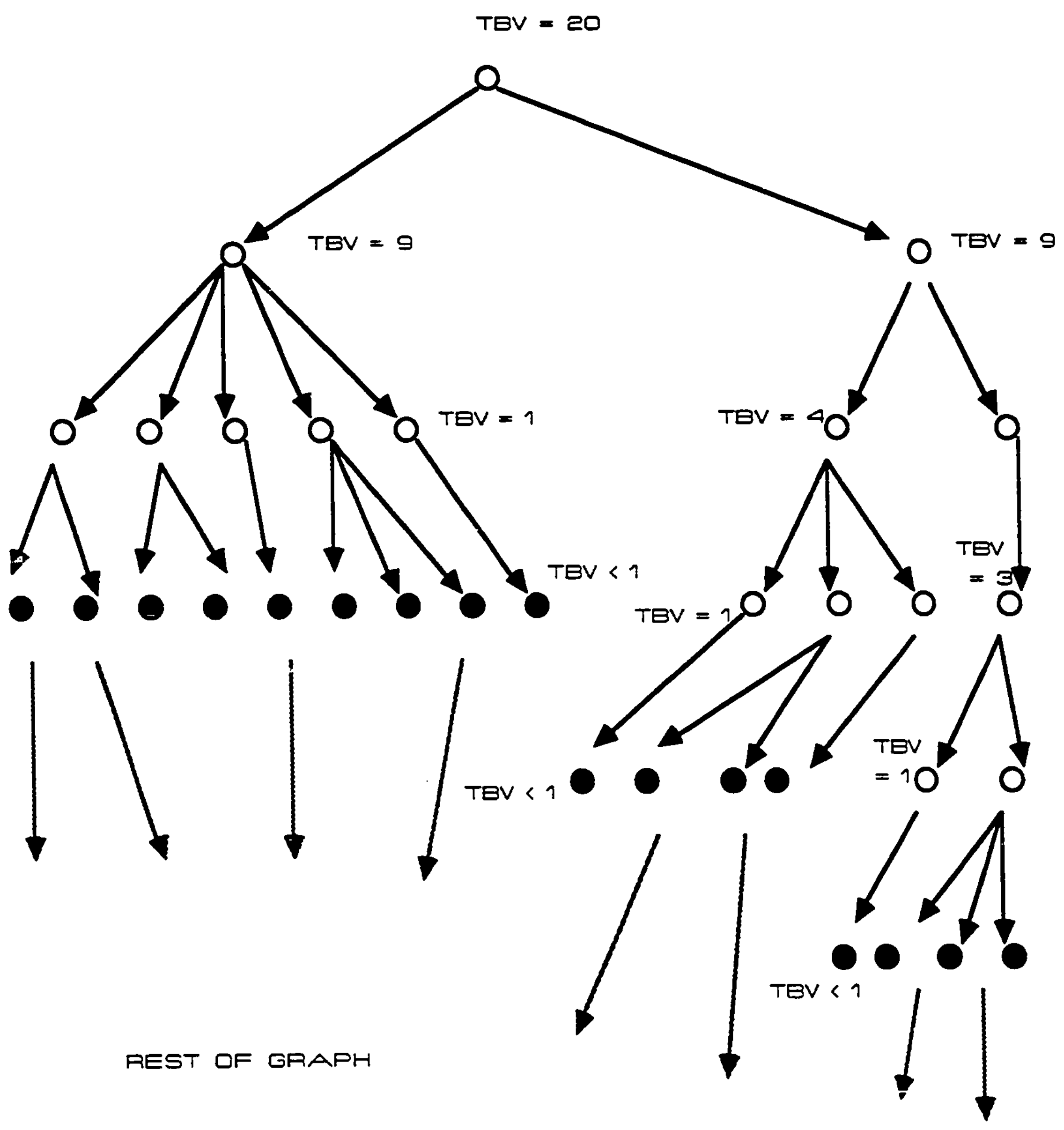
A depth first search (dfs) very often serves as a skeleton around which many efficient graph algorithms can be constructed and such a method will be used here with suitable modifications. The question which arises is to what level (or depth) should a dfs go ? The search space for a dfs grows exponentially with depth and even in fairly sparsely connected graphs this can lead to a combinatorial explosion which would slow down the system to an unacceptable degree. One feasible solution is to determine a ' reasonable ' depth based on empirical evidence obtained by implementing the system on available hardware. Besides being irrational and unduly restrictive, such a method leads to other problems. This is because in some cases , where branching is small , the traversal is done very quickly and only a few nodes are visited. In other cases , where branching is very extensive , the traversal has to visit a very large number of nodes and the time required is very long. Thus the space searched is very small in one case and very large in another. Since there usually is an upper bound on the time available for a search, the search depth is determined on the basis of a worst case of extensive branching. While this is fine for the worst case, it renders the system idle and unproductive when the branchng is low. This is because the system , in the time available to it , could have gone to a greater depth and could have done a better job of filtering out associated concepts.

Equibalance search is basically a depth first search but the maximum search depth along any particular search tree is determined dynamically as the search proceeds and could be different on different trees. To do so, the total number of nodes to be visited (TBV nodes) is determined beforehand and is supplied as a parameter to the search procedure. As a node is visited the TBV number is reduced by 1. This reduced number is divided by the number of children, rounded off to the nearest integer, and this gives the TBV number for each child. As long as the TBV number of each child is greater than 1, the search procedure is called recursively with each child. To avoid cycling, back arcs , that is arcs going from a node to a proper ancestor, are ignored. The actual search algorithm is as follows :

```
procedure  EBS  ( current_node  :  node_type;
                    TBV , search_level  : integer);

begin
visit  (current_node, search_level);
TBV :=  TBV - 1;
new_level  :=  search_level  + 1;
C  :=  set of all proper children of current_node;
n  :=  cardinality of set C;
new_TBV  :=  round ( TBV / n );
if new_TBV > 1
     then  for each  child node  in C
                do  EBS (child_node, new_TBV, new_level);
end;
```

Figure 2 shows the search tree generated by an equibalance search begun with a TBV number of 20.



NOTE : The search stops before visiting the the black nodes.

Figure 2 : An EBS search tree on a hypothetical graph

Size of Search

The size of the search, that is, the number of TBV nodes, is determined by the characteristics of the stored data together with the information needs of the user and the time available to perform the search. The basic database contains a very large number of concepts and it is expected that the user would want to view only a fraction of these which are closely related to his original query. Within a set of concepts, a measure of diversity is the average hamming distance. The hamming distance from concept A to concept B is defined to be the number of features in B which are not in A or vice versa. In a set of , say n, concepts we can calculate n^2 hamming distances and the average of these distances would represent a measure of the diversity of the set. This number would be large for the set of all concepts in the database, infact it would be close to the average number of features per concept, and it is expected that the set of concepts in the short-term-memory would have a smaller average hamming distance. This is because the short-term-memory is populated by concepts which are closely allied and have reletively more common features.

If the size of the search is set to a very high number, that is the initial TBV number is close to the total number of concepts in the database, then the search would eventually visit all the nodes, or concepts, and the average hamming distance of the short-term-memory would be equal to the average hamming distance of the complete database. Obviously this is pointless because nothing has been achieved even after spending a lot of CPU time. On the other hand if the TBV number is set equal to 1 then the short-term-memory would have just one concept and the average hamming distance would be equal to zero. Again this is a trivial solution to the problem because we were interested in fishing out a set of closely allied concepts and evidently this has not happenned. So the user must decide upon the degree of diversity, or closeness, that he requires to be present in the short-term-memory and determine an appropriate

TBV number for the search. Further, the time required for the search depends on the TBV number and so this could also be a factor in determining the size of the search.

Implementation and Results

A prototype system was coded in Waterloo pascal and implemented on the university IBM 4381 computer running VM/SP rel. 4. A set of 100 features was defined and 1000 concepts , each of size 7 , were generated by taking random combinations of the features. This constituted the database which was stored in the manner described in chapter 3. The average hamming distance of this set of concepts was 6.62, which is close to 7, the number of features in each concept.

Seven distinct concepts were chosen at random and the system was allowed to 'ramble' from each of these. In each case several different search sizes were specified and the results are shown in Table 2 and plotted in Graph 3. The mean of the seven runs is plotted in Graph 4 together with the two theoretical boundary cases discussed earlier in the section on search size.

Review of Results

The results were as expected and the average hamming distances of the set of concepts in short-term-memory increased rapidly with increase in search size. The only exception occurred in run 4 where there was a slight fall when the search size was increased from 40 to 60. To explain this we note that the system goes through a number of major cycles and each cycle is initiated by a pole concept. The pole concept is usually, but not necessarily, created by taking the features which occur most often in the concepts visited in the previous major cycle. Since the distribution of features among concepts is not completely homogenous, local perturbations of feature density affect the search. and could have caused this deviation from expected behaviour.

Also it has not been possible to test the system at search size = 2 for all seven starting concepts. This is because when the system is performing so close to the theoretical boundary. problems of singularity arise and cause runtime errors. The present program was not sophisticated enough to take care of all such cases and needs to be upgraded.

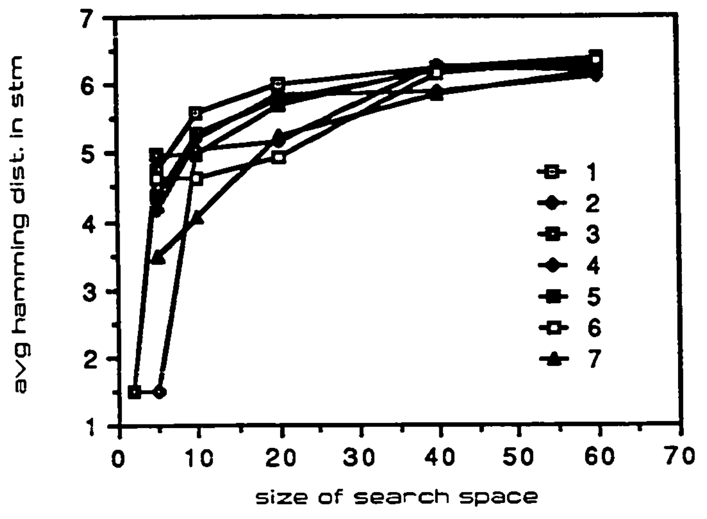
Barring these problems. the system performs very well and results are very much in agreement with theoretical expectations. From graph 4 we note that for the current data . a search size of 5 or 10 would be quite suitable though it is up to the user of the system to actally decide on this value.

size of search space

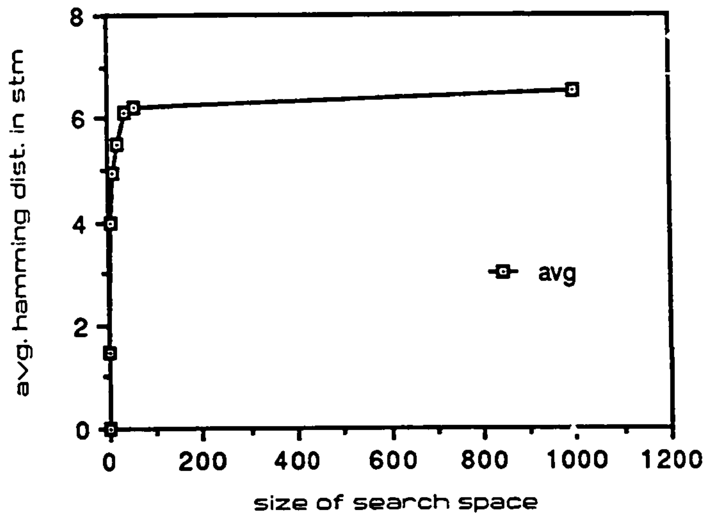
	1000	60	40	20	10	5	2	1
1		6.40	6.23	5.99	5.60	4.74	1.50	
2		6.12	5.91	5.87	5.19	4.16		
3		6.35	6.22	5.70	4.97	4.97	1.50	
4		6.19	6.25	5.15	5.04	1.50	1.50	
5		6.28	6.21	5.78	5.26	4.37		
6		6.33	6.17	4.93	4.62	4.62		
7		6.19	5.85	5.24	4.06	3.50		
average	6.51	6.27	6.12	5.52	4.96	3.98	1.50	0.00

note : values shown for sizes 1000 and 1 are from theoretical considerations.

Table 2 : Average hamming distance of concepts in STM.



Graph 3 : Average hamming distances of concepts in STM



Graph 4 : Mean of seven samples

Chapter 6

Associations of Physical Proximity

Associative information systems discussed so far have dealt with conceptual proximity. Pieces of information which were related to each other semantically, or otherwise in a contextual sense, were kept stored associatively. In this section, we look at associative data models which store information about objects which exist in close physical proximity. The rationale behind this approach is that once we show an interest in some object, it is likely that we would be interested in some neighbouring object. So once we access information about the former, it should be possible to get information about the neighbours easily. Information about neighbours, or at least its location, should be available within the information already accessed.

Paradigms of physical proximity are found in attempts to model spatial knowledge. A person's cognitive map, or knowledge of large scale space, is built up from observations gathered as he travels through the environment. Early work in this area (Tolman, 1948) consisted of investigations into 'field maps', but with the developmental work of Piaget et.al. (1960), many authors have recognized a distinction between cognitive spatial description and route maps. Nevertheless, most work, especially those dealing with explicitly computable models, have dealt almost exclusively with acquisition and use of knowledge of relative position.

Crane (1967) has discussed the possible advantages to be gained through use of associative memory in finding the shortest path through a large graph having unequal lengths and has given an algorithm which exploits the highly parallel search possible with associative memory.

Kuipers (1979) has presented a computational model of human commonsense knowledge of large scale space. Observations are assimilated into a description, from multiple perspectives, of the spatial environment. Kuipers (1983) discusses several different types of knowledge : of sensorimotor, topological and metrical spatial environment. One of the most interesting responses observed in trying to learn about route knowledge in humans is " I cant tell you how to get there, but I just know what to do (to get there) ".

When a human being traverses an unknown landscape, without a map in hand, he has an extremely local view of his surroundings. He follows instructions like " go straight until you come to a McDonalds " or "head towards the hills until you come to a river, then follow it". This can be modelled as follows.

Knowledge of City Maps

The city is represented as a network of roads. The portion of a road bet ween two junctions or between a junction and a dead end is a segment. Each segment has an arbitrary front eno and a back end. Both the ends of a segment are connected to the ends of other segments, except in the case of a dead end. This description can be easily implemented on a computer using records and pointers. The data structure used is shown in figure 2. This structure ensures that once access to information to one segment is established, it is very easy to access information about neighbouring segments. What is more important is that the time required to access the data on neighbours is independent of the total number of segments in the network. To test the efficacy of this data model, a simple heuristic algorithm is proposed which would allow a robot to traverse the map.

Associated with each segment is a 'picture' - which in the simplest case could be a binary matrix. The nature of the matrix would represent the position of the segment on the network. For example the matrix of the extreme south-west corner would be all zeroes and those of the extreme north-east segment would be all ones and intermediate matrices would be graded accordingly. More meaningful schemes could be tried as well. The robot is given a picture of the destination segment and let loose. The robot's view is extremely local. it can see only the pictures of the neighbouring segments. or at most the neighbour's neighbours upto a certain fixed depth.. Based on this view. it matches the pictures of the segements which it can see with the picture of the destination segment and in a 'greedy' manner. chooses a path which leads it to a segment which is closest to the destination. .

This method is fraught with dangers. Some roads might lead nowhere. or short cuts may be available after a short detour. However we are not currently interested in shortest path. These problems are a class by themselves with their own body of literature. Nevertheless this algorithm/datastructure has certain nice properties :

(a) it works in time independent of the total number of segments of the network. that is irrespective of the size of the map.

(b) it is intuitively similar to a human being walking across an unknown landscape. It makes the same type of mistakes which would be made by a human.

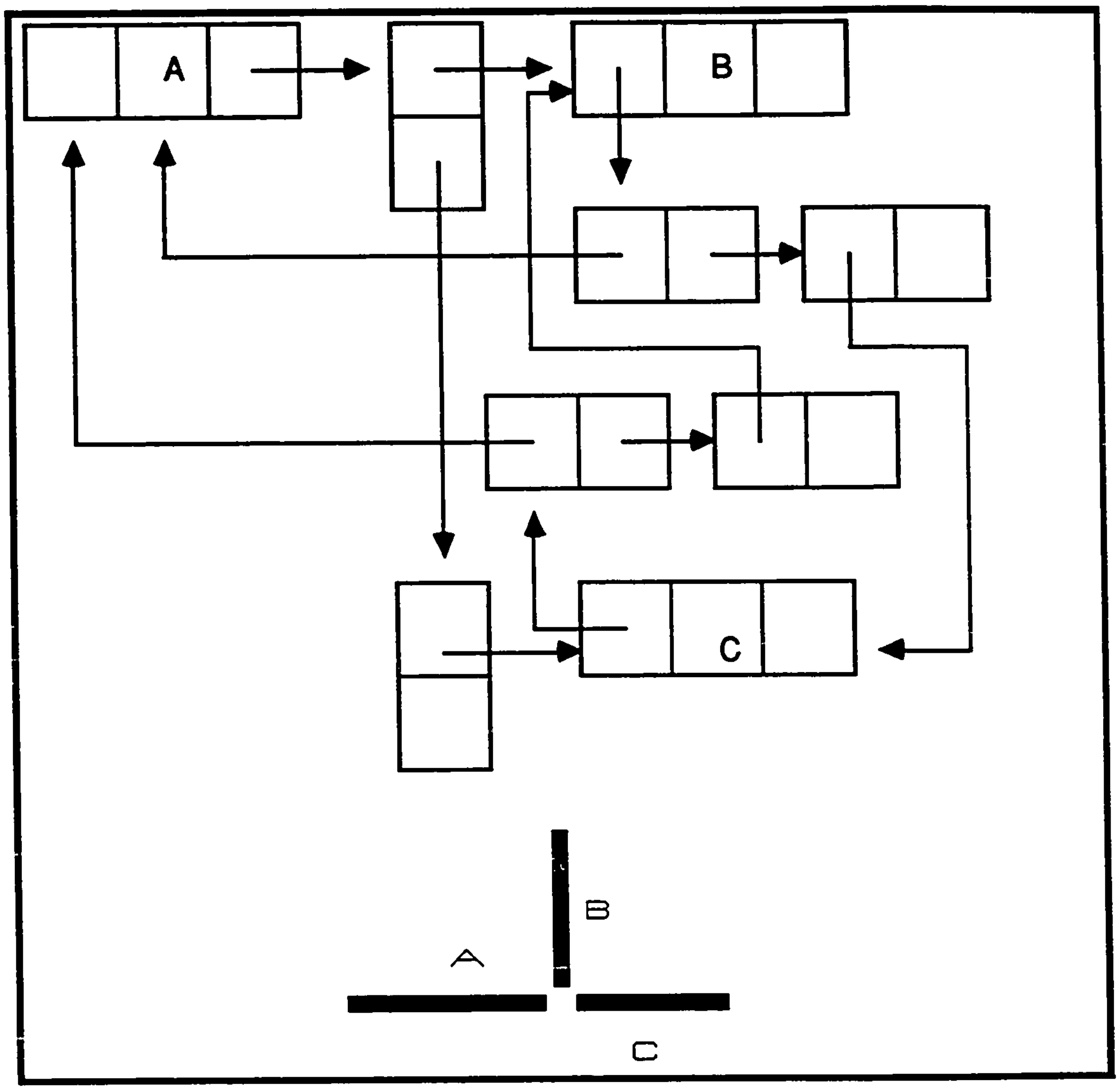


Figure 3 : Implementation of a three road intersection

Chapter 7

Conclusion

In this dissertation we have made an attempt to design an information system based on certain characteristics of human memory. Without debating the precise meaning of 'existence', we assume that in the human brain, memory co-exists with intelligence, but for our purpose we have chosen to model the memory aspect only. As the name suggests, this dissertation seeks to explore associations among various components of memory and utilize these to facilitate storage and retrieval of information.

Two types of associations have been explored , namely , logical and physical. While physical associations are interesting, they are very application specific and attempts to generalize them proved to be difficult. On the other hand logical associations can be easily identified in almost all types of information and a general theory of storage and retrieval based on this characteristic has been described in great detail in chapters 3, 4 and 5.

In the analyses carried out so far we have tried to keep away from all physiological considerations. However an important motivation for our model has been the structure of human memory and it is interesting , at this point , to try to relate the two. Hofstadter (1979) is . . . .

" . . led to the conclusion that for each concept there is a fairly well defined module which can be triggered - a module that consists of a small group of neurons - a neural complex . . A problem with this theory - at least if it is taken naively - is that it would suggest that one should be able to locate such modules somewhere within the brain. This has not yet been done, and some evidence. . . . points against localization. However it is still too early to tell. There may be many copies of each module spread around, or modules may overlap physically; both these effects would tend to obscure any divisions of neurons into packets. . . . It is even possible phenomena There are many questions that come to mind concerning these hypothesized neural complexes. For instance :

Do they extend into the lower regions of the brain ?
Can a single neuron belong to more than one complex ?
To how many complexes can a single neuron belong ?
By how many neurons can such complexes overlap ?
Are these complexes pretty much the same for everybody ?
Are corresponding ones found in corresponding places in different peoples brains ?
Do they overlap in the same way in everybody's brain ?

Philosophically the most important question of all this is : What would the existince of modules tell us ? Would this give us any insight into the phenomena of our own consciousness ? Or would it still leave us as much in the dark about what consciousness is, as does the knowledge that a brain is built out of neurons and glia ?"

We shall ignore the first and sixth questions posed by Hofstadter, which are of interest only to physiologists, and concentrate on the rest. In our model, single neurons do belong to more than one complex, and n the average number of concepts which share a feature (page 15) is same as the number of complexes to which a single neuron belongs. Our model puts no restrictions on the number of neurons by which complexes overlap. However we have carefully avoided the issue of nested concepts so that all the features in a concept are in general not shared by some other concept. The question of isomorphism - whether everybody views concepts and their relationships identically - is addressed by using fuzzy sets with weights attached to features in a concept. Different people would attach different weights to features in a concept but it is expected that if a feature is important most rational brains would give it a high weight. On the other hand if certain features are not very important some people would give them very low weights or they might fall below the threshhold, (page 28) and be ignored. This would lead to differences in the way complexes overlap in different brains.

As regards the final question, our model gives us a handle to attack the problem. It would be preposterous to claim that we have modelled consciousness. but we feel that we have taken a crucial first step from a low level - neuron by neuron - description to a high level - module by module - description.

Appendix

```
{ concept1 modified with the corrected insert procedure}
{ concept4 modified to incorporate delete etc...      }
{ now introducing remember & forget                      }
{ concpt5 modified ...                          }
{ this is a copy of concpt6 .. new name                }
{ fixing the error in function hash .. original stored in ver01 }
{ filter01 with a better remember procedure .....      }
{ improving the thinkabout procedure .. original in ver01  }
{ cleaned up version of filter02                        }
{ filter30 with space counter                           }
{ cleaned up version of filter31 .. and removing max_steps }
{ cleaned up version of filter40 .. any errors go back to 40 }
{ filter41 writing into a file }
{ final version of filter42 ... hopefully               }
{ filter50 with a different weighting function ... a ** -n }
{ filter60 modified to print all steps                  }
{ filter50/60 were erroneously counting some concepts which
  were not being visited but which could have been visited
  ... the error was only in counting not in the sequence
  of steps visited .. this has been corrected now       }
{ filter71 being changed to accomodate a new search strategy  }
{ filter71 being changed to accomodate a new weight assignment 1/logn}
{ think01 modified to take care of cycling ...          }
{ think02 suffers from stack overflow .. standard sets cannot be used}
{ modifying think02 with a better set structure ......  }
{ cleaned up version of think03  .. any errors go back to think04 }
{ set recursion errors of think04 being corrected here     }
{ some minor 'boundary' errors of think05 corrected here   }
{ think06 changed to record and writeout the reduced set   }


{$ stat=1000000000}
{$ ti=50000}

program concepts (input, output);

const  no_of_ideas   = 100;
       concept_size = 7  ;
       cs            = 8  ; { must be concept_size + 1}
       max_steps     = 5;
       max_level     = 10;
       max_avail     = 40;
       all_steps     = true;
       cut_off       = 0;
```

```
        max_con       = 1;   { confidence level _ sort of }
        data_file     = 'concept data2';
        results       = 'think08  output a';
        s_t_memory    = 'think08  stm a';


type    ideas     =  0 .. no_of_ideas;
        concepts =  1 .. 6000;
        concept  = record
          id      : integer;
           content : array [1 .. concept_size] of ideas;
          end;
        concept_token  = record
          id     :  integer;
          loc    :  ideas;
          end;
         x_concept = array [1.. concept_size] of ideas;
        point =      ^cell;
        cell  =  record
          id         : integer;
          prev_idea : ideas;
          next       : point;
          end;

        point_2   =  ^cell_2;
        cell_2    =  record
            id     : integer;
            loc    : ideas;
            count : integer;
            next  : point_2;
            end;

        point_4   =  ^cell_4;
        cell_4    =  record
            id      : integer;
            loc     : ideas;
            next    : point_4;
            end;

         con_set   =  ^node;
         node      =  record
             element : concepts;
            left     : con_set;
            right    : con_set;
             end;


 var  table                           : array [1..no_of_ideas] of point;
     launch_pad                        : array [1..11]    of x_concept;
     buffer                           : array [1..no_of_ideas] of real;
```

```
    terminus                        : x_concept;
    xx                            : concept_token;
    tag                           : point_4;
    reduced_set                     : con_set;
    new_buff              : array [1..max_steps ] of concept_token;
    last_concept,qq,          buff_end,last_pad   : integer;
    ans1,space, space_2, space_3                 : integer;
    outfile,out_2                 : text;

procedure print_banner;
begin
writeln (outfile,'**************************************************');
writeln (outfile,'concept_size        =  ',concept_size :4);
writeln (outfile,'data file          =  ',data_file);
writeln (outfile,'size of search      =  ',max_avail:4);
writeln (outfile,'search strategy     : dfs/constant # of nodes');
writeln (outfile,'weighting function =    1/level  ');
writeln (outfile,'maximum iterations =   ',max_steps:4);
writeln (outfile,'**************************************************');
end;



procedure insert_con (x : concepts; var A : con_set);
{
    procedure to insert a concept into a set of concepts
}
begin
 if A = nil then begin
                new (A); space_3 := space_3 + 1; A^.element := x;
                A^.left := nil; A^.right := nil;
                end
           else if x < A^.element
                   then insert_con (x,A^.left)
                   else insert_con (x,A^.right);
end;

procedure destroy (var A:con_set);
{
    destroys a set of concepts and returns the dynamic memory used
}
var B, C : con_set;
begin
if A <> nil then begin
                B := A^.left; C := A^.right;
                dispose (A) ; space_3 := space_3 - 1;
                destroy (B); destroy (C);
                end;
A := nil;
end;
```

```
procedure copy_set (original : con_set;
              var  copy     : con_set);
{
   duplicates a set of concepts
}
var  cc : integer;
procedure traverse (R : con_set);
begin
if R <> nil then begin
                 insert_con (R^.element,copy); cc := cc + 1;
                traverse (R^.left);  traverse (R^.right);
                end;
end;
begin
cc := 0; copy := nil;
traverse (original);
{writeln (cc :4, ' elements of the set copied');}
end;


function member (x : concepts; A : con_set) : boolean;
{
   returns TRUE if concept x is present in set A
}
begin
if A = nil
    then member := false
    else if x = A^.element
            then member := true
            else if x < A^.element
                    then member := member (x, A^.left)
                    else member := member (x, A^.right);
end;



procedure forget;
{
  destroys the list containing the concepts visited in one major cycle
}
var current,prev : point_4;
begin
if tag = nil then  writeln (outfile)
            else begin
                 current:= tag;tag :=nil; write (outfile,'forgetting');
                 while current <> nil do begin
                     prev := current; current:= current^.next;
                     write (outfile,prev^.id:4);
                     dispose (prev);  space_2 := space_2 -1;
                     end;
                 writeln (outfile);;
```

```
                  writeln (outfile,'these concepts visited earlier');
                 end;
end;


procedure remember (rm_token : concept_token);
{
  creates a list containing the concepts visited in one major cycle
}
var  current, new_tag,prev  :  point_4;
     add,  stop,first   :  boolean;
     x, y                :  integer;

begin
x := rm_token.id;  y := rm_token.loc;
if tag = nil
  then begin
       new(new_tag); new_tag^.id := x; space_2 := space_2 + 1;
       new_tag^.loc := y; new_tag^.next := nil;
       tag := new_tag;
       end
  else begin
       current := tag; prev := nil; stop := false;
       while not stop do
          if current^.id = x
              then begin stop := true; add := false;end
              else if  current^.id > x
                      then begin stop := true; add := true; end
                      else if current^.next <> nil
                              then begin
                                   prev := current;
                                   current := current^.next;
                                   end
                              else begin stop := true; add := true;
                                         prev := current; end;
       if add
          then begin
               new (new_tag); new_tag^.id := x; new_tag^.loc := y;
               space_2 := space_2 + 1;
               if prev = nil
                  then begin
                       new_tag^.next := tag; tag := new_tag;
                       end
                  else begin
                       new_tag^.next := prev^.next;
                       prev^.next := new_tag;
                       end;
               end;
       end;
 end;

 procedure delete (token : concept_token);
```

```
{
   deletes a concept from the database .. used to remove artificial
   concepts
}
var   start, j, jj         : ideas;
      i,k                  : integer;
      stop1, stop2          : boolean;
      current, prev         : point;

begin
i := token.id ; j := token.loc ; start := j ; k := 1 ; stop1 := false;
while not stop1 do begin
   current := table[j]  :   stop2 :=  false;  prev := nil;
   while not stop2 do begin
       if current^.id  = i
          then begin
               stop2 := true;  jj := current^.prev_idea;
                if prev = nil then table[j] := table[j]^.next
                           else prev^.next := current^.next;
               j := jj;
               end
          else begin
               prev := current;
               current := current^.next;
                if current = nil then begin
                                    stop2 := true;
                                    writeln (outfile,'error X');
                                    end;
               end;
       end;
   if j = start
      then stop1 := true
      else begin
           k := k + 1;
            if k > concept_size then begin
                                   stop1 := true;
                                    writeln (outfile,'error X1');
                                    end;
           end;
   end;
 end;

 procedure insert (i_con : concept);
 {
   a very basic procedure used to create the primary database ..
   inserts a concept into the database
 }
 var   current, new_point,prev    : point;
      i, name                  : integer;
      j, previous,first        : ideas;
       add, stop,first_time   : boolean;
```

```
begin
name := i_con.id;  first_time := true;
for i := 1 to concept_size do begin
  j := i_con.content[i];
  if j > 0 then begin
  if table[j] = nil
    then begin
         add := true; new(new_point);
         new_point^.next := nil; table[j] := new_point ;
         end
    else begin
         current := table[j]; add := true; stop := false; prev := nil;
         while not stop do
           if current^.id = name
             then begin
                  stop := true;add := false;writeln (outfile,'error01');
                  end
             else if current^.id > name
                    then begin stop := true; add := true; end
                    else if current^.next <> nil
                           then begin
                                prev := current;
                                current := current^.next;
                                end
                           else begin stop := true;add := true;
                                      prev := current; end;
         if add
           then begin
                new (new_point);
                if prev = nil
                   then begin new_point^.next := table[j];
                              table[j] := new_point; end
                   else begin new_point^.next := prev^.next;
                              prev^.next := new_point; end;
                end;
         end;
  if add
    then begin
         new_point^.id := name;
         if first_time
            then begin
                 first := j;  previous := j; first_time := false;
                 end
            else begin
                 new_point^.prev_idea := previous; previous := j;
                 end;
         end;
  end; { of IF}
  end; { of FOR}
 current := table [first];
 stop := false;
 while not stop do
```

```
  if current^.id = name
     then begin
          stop := true; current^.prev_idea := previous;
          end
     else begin
          current := current^.next;
          if current = nil
             then begin
                  writeln (outfile,'error02'); stop := true;
                  end;
          end;
end;

procedure load_data;
{
  reads the input file and creates the database
}
var con  : concept;
    x, i : integer;
    infile : text;

begin
for i := 1 to no_of_ideas do
    table[i] := nil;
reset  (infile, data_file);                  last_concept := 0;
while not eof(infile) do begin
  read (infile,x); con.id := x;
            if x > last_concept then last_concept := x;
  if (x mod 100) = 0 then writeln (x:5);
  for i := 1 to concept_size do begin
      read (infile,x); con.content[i] := x
      end;
  readln (infile);
  insert (con);
  end;
writeln (last_concept:6,' concepts read in');
end;

procedure load_launch_pad;
{
  reads in a set of data which is used to check the program
}
var i,j,x,ii    : integer;
    infile  : text;
    stop     : boolean;
    test_set  : set of concepts;
begin
{   test_set :=[20,210,230,310,410,530,610,630,810,930     ];   }
{  test_set :=[20,210,230,310,410                         ];   }
   test_set :=[            310                  ];
reset (infile,data_file);            ii := 0;
for i := 1 to last_concept do begin
```

```
    if (i mod 100) = 100 then write (outfile,i:5);
    read (infile,x);
    if x in test_set then begin
      ii := ii + 1;
      for j := 1 to concept_size do begin
          read (infile,x); launch_pad[ii][j] := x;
         end;
      end;
    readln (infile);
    end;
last_pad := ii;
end;


procedure display (token : concept_token);
(
  given a concept and one feature .. prints out all the other features
)
var start, j  : ideas;
    i,k       : integer;
    stop1,stop2 : boolean;
    A          : x_concept;
    current    : point;

begin
for i := 1 to concept_size do A[i] := 0;
i := token.id;  j := token.loc;   start := j;
k := 1;  stop1 := false;
write (outfile,'I',i:4,'I   ');
while not stop1 do begin
  A[k] := j; current := table[j] ; stop2 := false;
  while not stop2 do begin
    if current^.id = i
       then begin
            stop2 := true; j := current^.prev_idea;
            end
       else begin
            current := current^.next;
            if current = nil
               then begin
                    writeln (outfile,'k = ',k:3);
                    stop2 :=  true; stop1 :=  true;
                    writeln (outfile,'errorX2');
                    end;
            end;
     end;
  if j = start
     then stop1 := true
     else begin
          k := k + 1;
          if k > concept_size
             then begin
```

```
                  stop1 := true ; writeln (outfile,'error11');
                  end;
          end;
   end;
for i := 1 to concept_size do begin
   write (outfile,A[i]:3);
   end;
 writeln (outfile);;
end;

procedure readout;
(
scans the list of concepts visited in a major cycle and prints out
the number visited and if required gives their identities
)
var current : point_4;
    rd_token  : concept_token;
    nn        : integer;
begin
(writeln (outfile,' these concepts visited in the last step '); )
current := tag; nn := 0;
while current <> nil do begin
 (
  rd_token.id := current^.id;  rd_token.loc := current^.loc;
  writeln (outfile,current^.id:5);
  display (rd_token);
 )
  nn := nn + 1;
  current := current^.next;
  end;
writeln (outfile,nn:4,'  concepts visited in the last step');
 writeln (outfile);;
end;

function intersect ( x,y : point) : point;
(
  performs a set intersection to get a reduced set of common
concepts
)
var z, prev, temp, currx, curry   : point;
     first                         : boolean;

begin
currx := x;   curry := y;  z := nil;  first := true;
while (currx <> nil) and (curry <> nil) do begin
    if currx^.id = curry^.id
        then begin
              new (temp);
              temp^.id := currx^.id;
              temp^.prev_idea := 0;
              temp^.next    := nil;
             if first
```

```
              then begin
                   first := false;  z := temp;
                   end
              else prev^.next := temp;
           prev := temp;
           currx := currx^.next;
           end
       else if currx^.id < curry^.id
               then currx := currx^.next
                else curry := curry^.next;
  end;
intersect := z;
end;

procedure retrieve (A0  : x_concept ; var B : concept_token);
{
  given a set of input features ... identifies the concept
  which contains them ....
  this procedure sets the time complexity of the process
}
var S, S1   : point;
    stop, first   : boolean;
    i,j, con  : integer;
    A    : x_concept;

begin
for j := 1 to concept_size do A[j] := 0;
i := 1;
for j := 1 to concept_size do
   if A0[j] > 0  then begin
                     A[i] := A0[j];  i := i + 1;
                    end;
j := 1; con := 0; stop := false; first := true; S := nil;
while not stop do begin                                        { 3 }
  if A[j] > 0
    then begin                                                { 4 }
      if first then begin                                     { 5 }
                   S := table[A[j]];  first := false;
                  end                                         { 5 }
              else S := intersect (S, table[A[j]]);
      if S = nil
        then begin                                            { 6 }
          writeln (outfile,'no match found'); B.id := 0; stop := true;
          end                                                 { 6 }
        else begin                                            { 7 }
          if S^.next = nil
            then begin                                        { 8 }
              con := con + 1;
              if con > max_con
                then begin                                    { 9 }
                  writeln (outfile,' matches with ',S^.id:3);
                  B.id := S^.id; B.loc := A[j] ; stop := true;
```

```
                    end                                                  { 9 }
                   else begin  {10 }
                    j := j + 1;
                     if j > concept_size then begin                      { 11}
                        stop := true; B.id := S^.id ; B.loc := A[j-1];
                          writeln(outfile,'possbl match on ',j-1:4,'ideas');
                        end;                                            { 11}
                    end;                                                { 10}
                end                                                      { 8 }
              else begin                                                 {8a}
                j := j + 1;
                if j > concept_size  then begin                          {8b}
                  stop := true; B.id := 0;writeln (outfile,'error61');end;
                end;                                                     {8a}
            end;                                                        { 7 }
         end                                                             { 4 }
       else begin                                                        { 12 }
         stop := true;
         if S = nil
           then begin                                                    { 13 }
             writeln (outfile,'no match'); B.id := 0;
            end                                                         { 13 }
           else begin                                                    {13a}
             if S^.next = nil
               then begin                                                { 14}
                 writeln (outfile,'possible partial match'); B.id := S^.id;
                                                     B.loc := A[j-1];
                end                                                     { 14 }
               else begin                                                {15}
                 writeln (outfile,'no unique match'); B.id := 0;
                end;                                                    {15}
              end;                                                      { 13a}
         end;                                                           { 4 }
     end;                                                               { 3 }
   end;


   procedure sort_down ( var A : point_2);
   {
      ranks the neighbours according to their degree of overlap
   }
   var c1, c2          : point_2;
       t_id, t_count : integer;
       t_loc           : ideas;

   begin
   c1  := A;
   while c1^.next <> nil do begin
     c2 := c1^.next;
     while c2 <> nil do begin
       if c1^.count < c2^.count
         then begin
```

```
            t_id := c2^.id; t_loc := c2^.loc; t_count := c2^.count;
             c2^.id := c1^.id; c2^.loc := c1^.loc;  c2^.count :=c1^.count;
            c1^.id := t_id; c1^.loc := t_loc ; c1^.count := t_count;
            end;
     c2 := c2^.next;
     end;
   c1 := c1^.next;
   end;
end;

function neighbours (token : concept_token) : point_2;
{
  given a concept - this function returns a list of
  neighbouring concepts together with their degree of overlap
}
var hash_table                : array [1..200] of point_2;
    i,k                       : integer;
    j, jj, start              : ideas;
    stop, first               : boolean;
    current                   : point;
    head, tail                : point_2;

procedure hash (n, a : integer);

var  add               : integer;
     junk              : real;
     h_current, h_new  : point_2;
     h_stop            : boolean;

begin
add := n mod 200; if add = 0 then add := 200;
if hash_table[add] = nil
  then begin
       new (h_new);  h_new^.id := n;  h_new^.count := 1; h_new^.loc :=a;
       h_new^.next := nil;  hash_table[add] := h_new; space := space +1;
       end
  else begin
       h_current := hash_table[add];  h_stop := false;
       while not h_stop do begin
         if h_current^.id = n
            then begin
                 h_current^.count := h_current^.count + 1;
                h_stop := true;
                 end
            else begin
                 if h_current^.next <> nil
                    then h_current := h_current^.next
                    else begin
                          new (h_new); h_new^.id := n; h_new^.next :=nil;
                         h_new^.count := 1; h_new^.loc := a;
                          h_current^.next := h_new; space := space + 1;
                         h_stop := true;
```

```
                          end;
                      end;
                  end;
              end;
end;

begin ( of function neighbours)
for i := 1 to 200 do hash_table[i] := nil;

i := token.id;  j := token.loc;   start := j; k := 1;  stop := false;
while not stop do begin
   current := table[j];  jj := j;
   while (current <> nil) do begin
     if current^.id <> i
         then hash (current^.id,jj)
        else j := current^.prev_idea;
     current := current^.next;
     end;
   if j = jj then writeln (outfile,'error4 .. no change');
   if j = start
       then stop := true
        else begin
             k := k + 1;
              if k > concept_size
                  then begin
                        stop := true;  writeln (outfile,'error12');
                        end;
              end;
   end;

 head := nil;  tail := nil; first := true;
 for k := 1 to 200 do begin
    if hash_table[k] <> nil
        then begin
             if first
                  then begin
                         head := hash_table[k];  tail := head; first := false;
                         while tail^.next <> nil do
                            tail := tail^.next;
                        end
                  else begin
                         tail^.next := hash_table[k];
                         while tail^.next <> nil do
                            tail := tail^.next;
                        end;
              end;
     end;

  sort_down (head);
  neighbours := head;
  end;
```

```
procedure show_neighbours ( N : point_2);
{
  displays a list of neighbours
}
begin
if N = nil
  then writeln (outfile,'error6 … no neighbours')
  else begin
        writeln (outfile,' neigbour address (frequency)');
        while N <> nil do begin
             write (outfile,N^.id:4,N^.loc:4,'(',N^.count:2,')');
           N := N^.next;
           end;
        writeln (outfile);
        end;
end;

procedure flush_buffer;
var i : integer;
begin
for i := 1 to no_of_ideas do buffer[i] := 0;
end;


function power (mp, np : real) : real;
{
    mp raised to the power np
}
var temp : real;
begin
if (mp > 0) then temp := np * ln (mp)
              else temp := -9999.99 ;
if (temp > -35.0)     then temp   := exp ( np * ln(mp))
                       else temp   := 0;
power := temp;
end;


procedure store_ideas (token : concept_token; l : integer);
{
   as the EBS proceeds and visits a concepts the features
   in the concept a stored together with a proper weight
}
var  start, j,jj         : ideas;
      i,k                : integer;
      stop1, stop2          : boolean;
      current             : point;

function weight (lev : integer) : real;
var x    : real;
    y    : integer;
begin
```

```
{
y := -1 * lev;
x := power (2.0,y);
}
weight := 1 / lev   ;
end;

begin
i := token.id;  j := token.loc;  start := j;  k := 1;  stop1 := false;
while not stop1 do begin
  buffer[j] := buffer[j] + weight (l);
  stop2 := false;  current := table[j];
  while not stop2 do begin
     if current^.id = i
        then begin
             stop2 := true; j := current^.prev_idea;
             end
        else begin
             current := current^.next;
             if current = nil
                then begin
                     stop2 := true; stop1 := true;
                     writeln (outfile,'error X3');
                     end;
             end;
      end;
  if j = start
     then stop1 := true
     else begin
          k := k + 1;
          if k > concept_size
             then begin
                  stop1 := true; write (outfile,'error14');
                  end;
          end;
  end;
end;

procedure dump_trash ( TT : point_2);
{
  destroys the list of neighbours and returns the dynamic memory
}
var temp : point_2;
    x1, x2  : real;

begin
while TT <> nil do begin
  temp := TT^.next;
  dispose (TT);  space := space - 1;
  TT := temp;
  end;
end;
```

```
function count (TC : point_2) : integer;
{
  counts the number of still surviving children who are to be explored
}
var  c_temp  : point_2;
     n       : integer;
begin
n := 0; c_temp := TC;
while c_temp <> nil do begin
   n := n + 1;
   c_temp := c_temp^.next;
   end;
count := n;
end;

procedure show_set (Y : con_set);
{
  displays the contents of a set of concepts
}
procedure trav (YY : con_set);
begin
if YY <> nil then begin
                  trav (YY^.left);
                   write (YY^.element:4);
                  trav (YY^.right);
                  end;
end;
begin
write ('('); trav (Y); write (')');
end;

procedure think_about (old : concept_token;
                       level,avail  : integer;
                  old_visited      : con_set );
{
  the heart of the information filtering system ...
  performs an equibalance search into the network and filters out
  associated concepts
}
var T,T_head, current, tail : point_2;
    visited                 : con_set;
    prev                    : point_2;
    max,jump,i,kkk          : integer;
    sp1, sp2, sp3,sp4, sp7  : integer;
    n_avail, num_of_options: integer;
    stop                    : boolean;
    token                   : concept_token;


 begin
 if level <= max_level
```

```
  then begin
       copy_set (old_visited, visited);
        insert_con (old.id,visited);
       avail := avail - 1;
{     ..............................................
    some display ....
}
       if all_steps
          then begin
                for i := 1 to level do begin
                  write (outfile,'     ');
 write (          '     ');
                  end;
                 write (outfile,level:1,'.'); display (old);
 write (          level:1,'.');  writeln ('(',old.id:4,')');
                end;
{   ..............................................}
       if level > 1 then store_ideas (old,level);
        remember (old);
        if not member (old.id, reduced_set)
           then insert_con (old.id, reduced_set);
       T := neighbours (old);
{  ..............................................
   removing neighbours who have been visited before
}
       current := T;  prev := nil;
        while current <> nil do
             if member (current^.id,visited)
                then  if current = T
                         then begin
                              T := current^.next; dispose (current);
                              space := space - 1; current := T; end
                         else begin
                             prev^.next := current^.next;
                              dispose (current); space := space -1;
                             current:=    prev^.next; end
                 else  begin
                        prev := current; current := current^.next; end;
  {  ..............................................
    keeping only the 'closest neighbours and removing rest
  }
        jump := 0; current:=T;
         if current <> nil then begin max := current^.count;
                                       stop := false ; end
                           else stop := true;
        while (not stop) do
          if current = nil
            then stop := true
            else
             begin
               if current^.next = nil
                 then stop := true
                else
                begin
                 if current^.next^.count < max then begin
                     jump := jump + 1; max := current^.next^.count; end;
                if jump > cut_off then begin
                   stop := true; tail := current^.next;
                   current^.next := nil;
                   dump_trash (tail);
                   end;
                current := current^.next;
                end;
             end;
       T_head := T;
{     ............................................................}

        num_of_options := count (T);
        if num_of_options < 1
           then begin
                writeln (outfile,'error110'); n_avail := 0;
                end
           else  n_avail := round (avail/num_of_options);
        if n_avail >= 1
           then while (T <> nil) do begin
                  token.id := T^.id; token.loc := T^.loc;
                  sp7 := space_2;
                  think_about (token,(level+1),n_avail,visited);
                 T := T^.next;
                  end;
        dump_trash (T_head);
        destroy (visited);
        end;
end;

procedure new_concept (var B : x_concept);
{
  creates a new concept based on the most popular features
  encountered in the equibalance search
}
var i,j      : integer;
    max      : real;
    count : array [1 ..cs] of real;
    A        : array [1 .. cs ] of ideas;

begin

for i := 1 to concept_size + 1 do begin
   max := 0;
   for j := 1 to no_of_ideas do begin
      if buffer[j] >= max
         then begin
              A[i] := j;
              max  := buffer[j];
              end;
         end;
      buffer[A[i]] := 0;
      count[i] := max;
      end;
if count[cs -1] <> count[cs]
   then begin
        for i := 1 to concept_size do B[i] := A[i];
        end
   else begin
        for i := 1 to concept_size do begin
           if count[i] = count[cs] then B[i] := 0
                                   else B[i] := A[i];
          end;
        end;
end;



procedure identify ( A : x_concept;  var T : concept_token);
{
  fills up a buffer with any artificial concept created
}
var  B  : concept;
     jj : integer;

begin
retrieve (A,T);
if T.id = 0
    then begin
         writeln (outfile,'new_concept');
         B.id := last_concept + 1; last_concept := last_concept + 1;
         for jj := 1 to concept_size do
             B.content[jj] := A[jj];
   write    ('inserting this concept ....'); writeln (B.id:5);
   for jj := 1 to concept_size do write (B.content[jj]:4); writeln;
         insert (B);
         T.id := B.id;  T.loc := A[1];
         new_buff[buff_end].id := T.id;
         new_buff[buff_end].loc := T.loc;
         buff_end := buff_end + 1;
          if buff_end > max_steps  then writeln (outfile,'error55');
         end;
 end;
 procedure  clean_out;
 {
   cleans out the artificial concepts created in one run
 }
 var ii  : integer;
 begin
 if buff_end > 1 then write (outfile,'deleting artificial concepts');
 for ii := 1 to buff_end - 1 do begin
```

```
    delete(new_buff[ii]);
    write (outfile,new_buff[ii].id:4);
    end;
 writeln (outfile);;
end;


procedure drive  (start : x_concept;
             var finish : x_concept );
{
  uses a set of test data to check the system
  start is a starting pole concept and this procedure performs a
  sequence of major cycles until the system halts at the concept
  called terminus.
}
var  step,prev_concept,ii,sp11, sp12   : integer;
     dd                        : concept_token;
     stop                      : boolean;
     empty_set                   : con_set;
     been_here_before              : con_set;

begin
for ii := 1 to max_steps do begin
   new_buff[ii].id := 0; new_buff[ii].loc := 0; end;
buff_end := 1;
stop := false; step := 1; been_here_before := nil;
while not stop do
{ ..............................................................
   a major cycle
  ............................................................  }
   begin
   if step > 1 then prev_concept := dd.id;
   identify (start, dd);
   if step > 1
     then begin
          if member (dd.id,been_here_before) then begin
            stop := true;
              writeln (outfile,'STOP: cycling ...............');
            readout;
            end;
         end;
   if not stop then begin
        insert_con (dd.id, been_here_before);
        forget; writeln (outfile,'.......................');
        write (outfile,'step ',step:3,'  starting from '); display (dd);
        flush_buffer; empty_set := nil;
        think_about (dd, 1,max_avail,empty_set);
       new_concept (start);
       finish := start;
       step := step + 1 ; if step > max_steps then  begin
                              stop := true;
                               writeln (outfile,'STOP : max_steps ...');
```

```
                                    readout:          end:
      end:
   end:
{  ..........................................................
                                                  end of major cycle
  ..........................................................  }

end:

procedure write_reduced_set:
var xxx, vv  : integer:
    infile   : text:
begin
reset (infile, data_file);
while not eof (infile) do begin
  read (infile, xxx);
  if member (xxx, reduced_set)
     then begin
         write (out_2, xxx:5);
         for vv := 1 to concept_size do begin
             read (infile,xxx); write (out_2, xxx:4); end:
          readln (infile) ; writeln (out_2);
          end
      else readln (infile);
  end:
writeln  (out_2,'.............................................');
destroy (reduced_set);
end:

procedure pre_process:
{
    a little housekeeping ...
}
begin
     writeln (outfile);   writeln (outfile);
   writeln (outfile,'*******************************************');
   writeln (        '*******************************************');
end:


{
    THE MAIN PROGRAM BEGINS HERE ........
}
begin
rewrite (outfile,results);rewrite (out_2, s_t_memory); print_banner:
load_data: space := 0: space_2 := 0: space_3 := 0:
tag:= nil: ans1 := 1  ; reduced_set := nil:
while ans1 = 1   do begin
   load_launch_pad:
   for qq := 1 to last_pad do begin
      pre_process:
       drive (launch_pad[qq].terminus);
      identify (terminus,xx);
      write (outfile,'process terminates at'); display (xx);
     write_reduced_set;
     clean_out;
     end;
   ans1 := 0;
   end;
end.
```

References

Anderson, J.A., Bower, G.H., (1974). *Human Associative Memory*, Wiley : New York.

Atkinson, R. C. , Shiffrin, R. M., (1968). Human Memory : a proposed system, in *The Psychology of Learning and Motivation*, (eds. K.W.Spence and J.T. Spence), Vol. 2, pp. 90 - 195, Academic Press: New York.

Banerji, R. D. (1962). The Description List of Concepts. *Comm. of the ACM*., Vol. 5, No. 8, pp. 426 - 432.

Bierwisch, M. , (1970). Semantics, in *New Horizons in Linguistics*, (ed. J. Lyons) , Penguin : Baltimore.

Blair, D. C. , Maron, M. E. (1985). An evaluation of retrieval effectiveness. *Comm. of the ACM*, Vol. 28, No. 3, pp. 289 - 299.

Boden, M. (1979). The Computational Metaphor in Psychology, in *Philosophical Problems in Psychology*, (ed. N. Bolton), Methuen : London.

Bower, G.H. (1975). Cognitive Psychology : An Introduction, in *Handbook of Learning and Cognitive Processes*, (ed. W.K.Estes), vol. 1.

Civanlar, M. R. , Trussel, H. J. (1986). Constructing Membership Functions using Stastical Data, *Fuzzy Sets and Systems*, Vol. 18, pp. 1 -13.

Codd, E. F. (1970) A relational model for large shared databases. *Comm of the ACM*, Voi. 13, No. 6, pp. 377 - 387.

Conklin, J. (1987). Hypertext : in introduction and survey. ***Computer***, Vol. 20, No. 9, pp 17 - 41.

Crane, B. A. (1967). Path finding with associative memory. IEEE Trans. on Computers. Vol. xx, No. xx

Dubois, D., Prade, H. (1980). ***Fuzzy Sets and Systems : Theory and Applications***, Academic Press : New York.

Feldman, J.A. , (1981) A Connectionist Model of Memory, in ***Parallel Models of Associative Memory***, (eds. G. E. Hinton and J. A. Anderson), Lawrence Erlbaum : Hillsdale.

Giles, R. (1988). The Concept of Grade of Membership. ***Fuzzy Sets and Systems***, Vol. 25, pp. 297 - 323.

Gilhooly, K. J. (1982). ***Thinking***, p. 150, Academic Press: New York.

Gillmartin, A., Newell, A., Simon, H.A., ((1976). A Program Modelling Short Term Memory under Strategy Control, in ***The Structure of Human Memory*** (ed. C. N. Cofer), Freeman : San Francisco.

Hanlon, A. C. , (1966). ***IEEE Trans. Electron. Comput.*** Vol. 15, pp. 509-521.

Hinton, G.E., Anderson, J.A. (1981). ***Parallel Models of Associative Memory***. Lawrence Erlbaum : Hillsdale, N.J.

Hinton, G. E. , (1981). Implementing semanting networks in parallel hardware, in ***Parallel Models of Associative Memory***. (ed. G. E. Hinton and J. A. Anderson), Lawrence Erlbaum : Hillsdale.

Hofstadter. D. R. (1979). ***Godel, Escher, Bach : an Eternal Golden Braid.*** Basic Books : New York. pp. 348 - 349.

Hunt. E. (1971). What kind of Computer is Man ?. ***Cognitve Psychology.*** Vol. 2. pp 57-98.

Hunt. E. (1973). The memory we must have . in ***Computer Models of Thought and Language.*** (eds. R.C. Schank and K.M. Colby). pp. 343 - 371. Freeman : San Francisco.

Katz . J. J. . (1972). ***Semantic Theory.*** Harvard University Press : Cambridge.

Kent. W.. (1983). A simple guide to the five normal forms in Relational Database theory. ***Comm. of the ACM,*** Vol. 26. No. 2. pp. 198 - 203.

Koestler. A. (1964). ***The act of creation.*** Hutchinson : London.

Kohonen. T.. Oja. E. . Lehtio. P. (1981). Storage and Processing of Information in Distributed Associative Information Systems. in ***Parallel Models of Associative Memory*** . (ed. G. E. Hinton and J. A. Anderson). Lawrence Erlbaum : Hillsdale.

Kohonen. T. (1984). ***Self Organization and Associative Memory.*** Springer Verlag : Hiedelberg.

Kolodner. J. L.. (1983a). Maintaining Organization in a Dynamic Long Term Memory. ***Cognitive Science.*** Vol. 7. pp. 243 - 280.

Kolodner. J. L.. (1983b). Reconstructive Memory : a computer model. ***Cognitive Science.*** Vol. 7.. pp. 281 - 328.

Kuhns. J. L.. Maron. M. E. (1960). On Relevance. Probabilistic Indexing and Information Retrieval . *Journal of the Association of Computing Machinery*. Vol. 7. No. 3. pp. 216 - 244.

Kuipers. B. (1979). Common Sense Knowledge of Space : Learning from Experience. *Proc. 6th. Intl. Conf. on Artificial Intelligence*. pp.499 - 501. Tokyo.

Kuipers. B. (1983). Modelling human knowledge of routes : partial knowledge and individual variations. *Proc. Natl. Conf. on Artificial Intelligence*. Vol. 1. pp. 216 - 219.

Landauer. T. K. (1986) How much do people remember ? Some estimates of the quantity of learned information in long term memory. *Cognitive Science*. Vol. 10. pp 477 - 493.

Lenk. P. J.. Floyd. B. D. . (1988). Dynamically updating relevance judgments in probabilistic information systems via user's feedback. *Management Science*. Vol. 34. No. 12. pp. 1450 - 1459.

Lindsay. P. H.. Norman. D.A. (1977). *Human Information Processing*. Academic Press : New YOrk. p. 421.

Linneman V. (1986) Constructorset's database support for knowledge based systems. *Proc. of the Intnl. Conf. on Data Engg.*

Malone. T.W. . Grant. K. R. . Turbak. F. A. (1986). The Information Lens : An intelligent system for information storing in organizations. *Proc. CHI '86 Conf. on Human Factors in Computing Systems*. Boston.

Maron. M. E.. (1982) Associative Search Techniques versus Probabilistic Retrieval Models. *J. Amer. Soc. Information Sci.*. Vol. 33. pp.308-310.

McClelland. J. L.. Rumelhart. D. E.. (1980/81). An Interactive Activation Model of the Effect of Context in Perception I and II. *Psychological Review.* Vol. 88. No. 5. Vol. 89. No. 1.

McClelland. J. L.. Rumelhart. D.E.. (1986). *Parallel Distributed Processing.* MIT Press : Cambridge. Mass.

Mednick. S. A. (1962). The associative basis for the creative process. *Psychological Review*. Vol. 69. pp. 220 - 232.

Minsky. M.. Papert. S. . (1969). *Perceptrons.* MIT Press : CAmbridge.

Minsky. M.. (1977). Plain talk about neurodevelopmental epistemology. *Proc. of the 5th intntl. conf. on artificial intelligence,* Boston.

Minsky. M.. (1980). K-Lines : A theory of memory. *Cognitive Science.* Vol. 4. pp. 117 - 133.

Morton. J.. (1970). A Functional Model of Memeory. in *Models of Human Memory.* (ed. D. A. Norman). Academic Press : New York.

Narayanan. A. . (1984). Memory Models of Man and Machine. in *Artificial Intelligence : principles and applications* . (ed. M. Yazdani). Chapman and Hall : New York.

Osherson. D. N.. Smith. E. E.. (1981). On the adequacy of prototype theory as a theory of concepts. *Cognition*. Vol. 9. pp 35 - 58.

Parsaye. K. (1983). Logic Programming and Relational Database. ***Database Engg.*** Vol. 6. No. 4. pp 202 - 211.

Piaget. J. B. . Inhelder. B.. Szeminska. A. (1960). ***The Child's Conception of Geometry.*** Basic Books : New York.

Quillian. M. R.. (1968). Semantic Memory. in ***Semantic Information Processing.*** (ed. M . Minsky). MIT Press : Cambridge.

Radecki . T. (1979) Fuzzy set theoretical approach to document retrieval. ***Inform. Process. Mgmt.*** . Vol. 15. pp. 247 - 259.

Rosenblatt. F. (1958). The Perceptron : A probabilistic model for information storage in the human brain. ***Psychological Review.*** Vol. 65. pp 368-408.

Rosenblatt. F. (1962). ***Principles of Neurodynamics.*** Spartan. : Washington D.C.

Schank R.. (1980) . Language and Memory. ***Cognitive Science.*** Vol. 4. pp. 243 - 284.

Schank. R.. (1982). ***Dynamic Memory.*** Cambridge University Press : New York.

Siefert. C. M. . McKoon. G. . Abelson. R. P. . Ratcliff. R. . (1986). Memory connections between thematically similar episodes. ***Journal of Expt. Psych. : Learning , Memory and Cognition.*** vol. 12. No. 2. pp 220 - 231.

Shasha. D. (1985) Netbook - a data model to support knowledge exploration . ***Proc. of VLDB ' 85.*** Stockholm. pp. 418 - 425.

Shasha, D. (1986). When does nonlinear text help ? *Proc. of the First Intnl. Conf. on expert databse systems* , (ed. L. Kerschberg).

Simon, H. A. (1979). *Models of Thought,* Yale University Press : New Haven, Conn.

Smith, E. E. , Medin, D. L., (1981). *Categories and Concepts.* Harvard University Press : Cambridge.

Smith, J. M. (1986). Expert Database Systems : A database perspective. *Expert Databse Systems : Proc. of the First Intnl. Workshop.* (ed. L Kerschberg). Benjamin/Cummings : Menlo Park.

Srull, T. K. , Lichtenstein, M., Rothbart, M. , (1985). Associative Storage and Retreival Processes in Person Memory. *Journal of Expt. Psych. : Learning , Memory and Cognition,* Vol. 11, No. 2, pp. 316 - 345.

Tollman, E. C. (1948). Cognitive Maps in Rats and Men, *Psychological Review,* Vol. 55, pp. 189 - 208. (reprinted in *Image and Environment,* R.M.Downs and D. Stea (eds.), Aldine : Chicago, 1973).

Ullman, J. D. (1980). *Principles of Database Systems,* Computer Science Press : Potomac.

Zadeh, L. ,(1965). Fuzzy Sets. *Information and Control,* Vol. 8, pp. 338 - 353.

Zadeh, L. ,(1982). A note on prototype theory and fuzzy sets. *Cognition,* Vol. 12, pp 291 - 297.

Zenner, R. B. R. C. , De Caluwe, R. M. M., Kerre, E. E. (1985). A new approach to information retrieval systems using fuzzy expressions. *Fuzzy Sets and Systems*, Vol. 17, pp. 9 – 22.

Vita

Prithwis Mukerjee was born in Calcutta, India, the son of Subhrendu and Reba Mukerjee on October 15, 1961. He attended St. Xavier's Collegiate School, Calcutta and passed the Indian Certificate of Secondary Education Examination in 1977. After completing the first year of the West Bengal Higher Secondary course in St. Xavier's College, Calcutta he joined the Indian Institute of Technology, Kharagpur in 1979 and graduated with a B.Tech (Hons) in Mechanical Engineering, in 1984. He spent a year in graduate study at IIT, Kharagpur and joined the University of Texas at Dallas in fall 1985. He received an M.S. in Management and Administrative Science with a concentration in Operations Research from UT-Dallas, in fall 1988.

He was married to Indira Bandyopadhyay of Calcutta on March 9, 1989.

His permanent address is :

"Renu Villa",
51 Jatin Das Road,
Calcutta 700 029, INDIA. .