Information Technology: Research and Development 1982), *1* (55-72)

THE FACT DATABASE: A SYSTEM USING GENERIC ASSOCIATIVE NETWORKS

D. R. McGREGOR AND J. R. MALONE

Department of Computer Science, University of Strathclyde, Glasgow, UK

(Received 6 February 1981, revised 17 August 1981)

ABSTRACT

A novel type of associative network, directly implementable as an electronic circuit, can provide flexible access to a database held as a semantic network. The circuit is used to expand the range of terms accessed by a query, and to access information by deduction by using set membership linkages which can be represented within it. The paper describes the main characteristics of the database, its data structure, and its most important operations. Software simulations of the system have been implemented. Initial performance results are reported.

1. INTRODUCTION

Information systems have two major problem areas. The first and perhaps more fundamental concerns the symbolic representation of information in such a way that it can be checked, inserted, updated, retrieved and acted upon whenever required. For example, few present-day systems can handle conditional or qualified information adequately. The second is that many collections of information of interest as databases are very large. Even though we may in principle have a method of doing the processing required it may nonetheless be impractical on computers with today's architecture. These limits are placed mainly by the relative inaccessibility of information stored on direct-access magnetic discs, and the serial nature of processors of the general Von Neumann type. This paper is a description of our efforts in these two areas.

The impetus for our new approach came from our earlier work on special-purpose processors and backing-stores. In this we developed and tested a fast microcoded vector processor suitable for handling highly repetitive database manipulations — sorting, merging, joining, as well as multi-term selection of data and functions for finding max, min count, etc. — provided data were resident in a RAM store accessible to the processor (McGregor *et al.*, 1976). We also investigated the performance of a disc controller with its own integral cache to contain recently-accessed data (Bagshaw and McGregor, 1980). Despite quite impressive gains in

performance in linear operations such as searching, we found — as have others (Lea and Schuegraf, 1980) — that purely serial operations are still much too slow to be used exclusively. The options available are either to abandon secondary storage of a passive type or attempt to build a data structure to locate the required data. For machines of a quasi-conventional architecture, therefore, what is required is a compact, indexable concentration of data, so that for any particular operation we can avoid dealing with the major portion of the database, and thus minimise the bottleneck of backingstore accesses.

Why should we strive towards a better representational model than those commonly adopted in database systems? First, we know from other fields of human endeavour that a more elegant model may greatly improve our understanding of a system. Arabic numerals, the Copernican model of the solar system and the structural formulae of organic chemistry are examples of such elegant representations. The last is particularly apt. Each structural formula represents a vast accumulation of experimental evidence in a form which can be used in planning and prediction in a way which would be quite impossible from a crude tabulation of the original data. Thus a more elegant representation can improve our ability to exploit a system.

In databases an inadequate data model means that we have to rely on human intuition and adaptability in querying the database. Some operations are valid whereas others — apparently similar as far as the data model is concerned — are invalid (or at best have a totally different meaning). An example may be taken from the *n*-ary relational model which permits the user to imagine a larger number of semantically meaningful relationships than actually exist in the system being represented. Consider the simple relation:

Person	*Project*	*Sex*
Smith, J	building	male
Jones, J	accounts	female

Projecting the domains on person/sex returns a relation which gives the sex of each person. The similar operation of projecting on project/sex certainly does not give the sex of each project! This is the well known 'connection trap' (Codd, 1970).

Where information must be supplied by *ad hoc* means — by the casual end user or built into specific applications programs — we are dealing with an inadequate data-model.

As we move on to database systems which interact with other computer systems without any human intervention, and databases which carry out inference automatically, the data model's correctness becomes paramount. Clearly more detail is required in the data model than is supplied by the simple *n*-ary relational model in the example above.

The work described here is a link between two fields of endeavour. Precise semantic networks have been used for some time by workers in artificial intelligence, but so far this approach has been largely ignored by workers in the database field. We believe that this is because up till now the general semantic net approach has seemed prohibitively expensive in storage and computational requirements for all but small database systems. A semantic net structure, making use of sets and generic information, is capable of supporting a more flexible and more powerfully descriptive database system than present relational and network models, and it is our purpose here to show that such an information structure can be implemented

with reasonable efficiency (even for large volumes of information) provided appropriate hardware or data structures are adopted. The work described here is part of an investigation into a novel database architecture. The new architecture (which is derived by extending the Binary Relational model) can be described in terms of its main data structure (built from units of information called 'Facts'), and a conceptual machine which can manipulate the structure (the Fact machine). The system is based on networks of generic information. In this paper we describe:

1. How information concerning sets and set memberships plays a key role in ensuring the system presents a flexible, 'friendly' — but precise — interface to the user.
2. Given the retrieval mechanism, how the 'secret' internal data structure can be reorganised to give a more compact and higher level representation for the database information, without the changes becoming apparent to the users of the system.
3. How the required generic associative network can be implemented.

Other aspects of the Fact system have been described elsewhere (McGregor and Malone, 1980).

2. THE DATA STRUCTURE

The data structure is constructed from Entities and Facts.

2.1 Entities

Following Senko (1975) we define an entity to be any distinct 'thing' or concept. This can be concrete or abstract, specific or general. When we wish to represent an entity from the world which we are modelling, we use a uniquely distinctive entity symbol within the database, which is always in one-to-one relationship with the real-world entity. For brevity, in the following discussion we shall frequently refer to entity symbols simply as 'entities'.

2.2 Facts

A 'Fact' is a named directed association between two entities. It is a 'molecule' of knowledge (and information) and is the smallest unit of structured knowledge which can have an independent existence, and has the quadruple structure with the fields named as shown below:

Fact Number	Subject	Relation	Object

It is, more properly, an assertion, in that it is possible to store untrue, doubtful or qualified information in this form. However for historical reasons and brevity, we find it easier to use the term Fact in a restricted sense to mean an assertion

represented by the method used here. Each Fact has a unique identity represented by its Fact Number, which also belongs to the set of entities. We refer to the two associated entities as the Subject and Object. The name of the association is called the Relation. The whole Fact is itself an entity, and when information about the Fact has to be represented in the database, this is done by using its unique Fact Number. Each Fact is unique, two Facts can have identical entities in the Subject, Relation and Object fields and still represent different information as the Fact numbers are distinct. Using the Fact Number to represent a Fact (in the Subject, Object or Relation fields of another Fact) it is possible to represent knowledge molecules of arbitrary complexity.

3. THE RETRIEVAL PROBLEM

Consider a database storing information about, say, employees. The typical database responds when queries are expressed in the general term 'employee', which corresponds to a domain name in its schema, e.g.,

For employee list manager.

or in terms of the specific entity:

For employee = 'John R. Smith' list manager.

However, the typical database will fail to return data at all to what to the outsider are equally specific or even more specific terms, e.g.,

For plumber list manager.

or

For salesman = 'smith' list manager.

This occurs because the intermediate level set names (plumber and salesman) are not treated as part of the more general super-sets (employee or person). This problem is common to almost all contemporary database systems.

One attempt to bypass the problem is to throw the burden of adaptation entirely on to the user. He or she must use the limited terms defined by the standard schema — in document retrieval by the standard indexing scheme or thesaurus. This works where the users are relatively expert and the schema is simple and stable. It is not suitable for casual users or where the schema is large, complex or subject to frequent change. Even more important, it is not going to work where the user is another machine, lacking the human adaptability on which this approach relies. In the future environment of decentralised systems and long-range communication systems some of the tasks of adapting to the terms used in requests for data must be undertaken by the database system itself.

The problem is not much alleviated by the common strategem of allowing less precise character matching based on 'variable length don't care' symbols. The obvious method of explicitly storing a multiplicity of denotations for individual entities and explicit additional set linkages soon becomes prohibitively expensive in storage space.

If, however, the system can handle 'recursive sets' or 'semi-explicit sets' (Fahlman, 1979) we can achieve flexibility with economy of storage for the cost of some additional processing. The concept is simple. Each entity can be a member of any number of sets, and sets can be members of other sets. When a set has members which are also sets, members of these too are taken to be members of the original set. Hence our use of the term 'recursive set'. Normally in mathematics a set is represented either explicitly, by enumeration, or implicitly — by the set of values which can be returned by a function. Recursive sets are used by our database system is represented in an intermediate way, the first level of membership of a set is enumerated explicitly in storage, but subsequent levels are obtained by manipulation of this stored information. Hence Fahlman's use of the term 'semi-explicit set'.

3.1 Use of generic information in retrieval: The Broom Concept

Information about set membership of entities is represented within a Fact database — and is acted on directly by the database system in a number of ways:

1. A query may use a denotation corresponding to a set of entities, e.g.,

   'schoolteacher', 'employee', 'person', 'lecturer'

   but the desired information in the database may be stored at a different level. For instance it may be stored at the level of the individual set members. We therefore need to 'broaden' the terms used in the enquiry to ensure that all relevant information in the database is accessed irrespective of the level of generalisation at which it is stored. A query about 'person' should be concerned with any subsets of person such as 'employee', and *their* subsets ('lecturers'), etc. To satisfy this requirement, the 'broadening' which we apply to the denotation used in the query is to attempt to retrieve any member of the powerset of the term supplied (Fig. 1).

2. There is also the possibility that information is stored 'higher up' the generic hierarchy than the term used in the query. For example, if all people are mortal, it would be entirely reasonable for the database to store this information with the entity representing the set of all people. On retrieval, we require that queries requesting information about subsets (lecturers, schoolteachers, chemistry lecturer, etc.) or particular members of those subsets — 'Fred Jones is mortal?' — should elicit the stored information. This requires simple deduction. In order to do this, the query terms must therefore be 'broadened' upwards so that the stored information is accessed, at whichever higher set it happens to be stored. We call this the 'upward closure' of the search term. Starting from the search term (which we refer to as the Principal Node) it includes all sets of which the Principal Node is a member, all sets containing any of those sets as a member . . . and so on until no additional sets can be accessed (Fig. 1).

The entire 'broadened' collection of search terms can be visualised as a 'Broom', consisting of the Principal Node, its Downward Closure — the 'Brush', and its Upward Closure — the 'Handle'.

To ensure that all relevant information is accessed the Fact System thus broadens

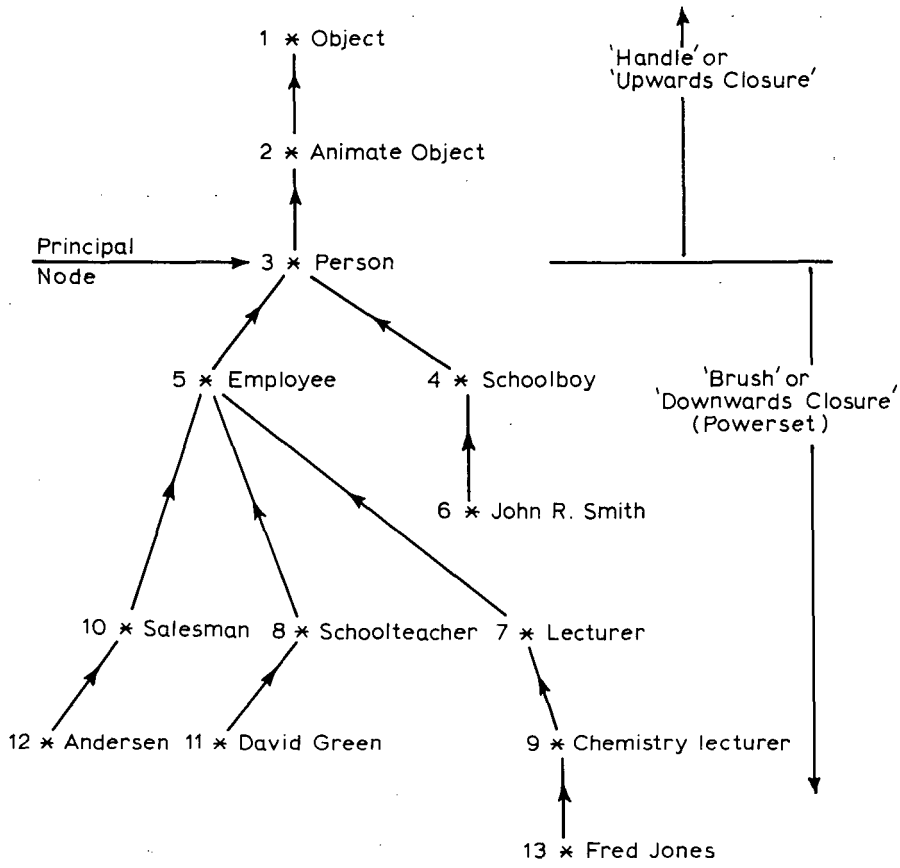

FIG. 1. 'Broom' of 'person'

each search term to the corresponding Broom, but selects information only if it lies on the intersections of all required Brooms.

The importance of this procedure is that:

1. It provides a precise but flexible interface with the outside world. The matching procedure seems to closely mirror the corresponding human process, but is convenient for both computer systems and human users.
2. It provides the system with the capability of restructuring its data, factoring common items and storing them higher up the generic structures, yet allowing users to access that information in a completely unchanged manner. We believe that this procedure is fundamental to any advanced database system.

4. AUTONOMOUS ACTIVITIES

Given the above retrieval mechanism, the flexibility of the data representation can be exploited by a set of background, or 'autonomous activities' concerned with organising the secret internal data structure. They include:

1. Clustering the data into additional high-level sets ['Logical Clustering'].
2. Simplification of the database, when two different entity symbols are found to be representing the same real-world entity.
3. Formation of simple hypotheses.

All three areas have given encouraging results in our investigations so far.

4.1 Creating additional generalised entities

It is possible to save a substantial fraction of the space occupied by an unfactored database, by collecting the entities into sets, then storing the common Facts at the set level. The properties of individual set members can be reconstructed when needed by the process of simple deduction. This is carried out automatically as part of the 'inferential fetch' mechanism.

The problem is to cluster the Facts in such a way that we obtain a useful generalised description of the low-level data model. This aim is compatible with a sub-goal — the saving of storage space. Whatever we do must always result in a system which can still reconstitute the original data, when required. The general clustering operation is illustrated in Figure 2.

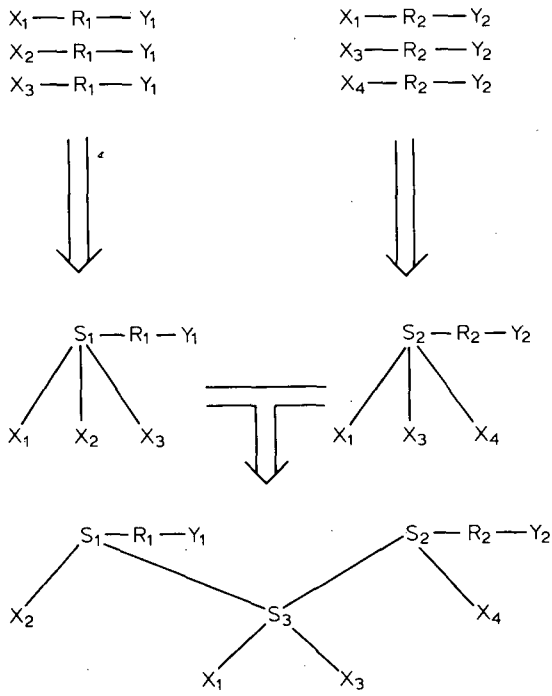

FIG. 2. Sequences in a clustering operation

New entities S_1, S_2, S_3 . . . . . are created to represent the largest sets with the largest number of attributes. In order to achieve a significant saving of space a minimum threshold must be applied, so that a new set entity is not created unless it has a significant number of members all of which share a number of attributes in common.

4.2 Description of the clustering method

All the Facts are sorted into an arbitrary order by the values of the Entity numbers in each of the Fact fields. We ourselves use a precedence for the three fields: Relation > Object > Subject.

The objective of the sorting operation is to locate similar Facts together, so that a simple scan can separate sets having Object and Relation fields in common. Where more than the minimum number of Facts are in a set, a new entity symbol is created to represent the set. New Facts are constructed to enable the set entity to represent the common attributes of the set, and to link the various members to the new set-entity. Finally the original Facts, now no longer required, are removed. The second stage of the clustering process is to compare all sets with each other, two at a time. If the intersection of two sets is greater than some minimum threshold new entities representing the intersection sets are generated, and the data structure is adjusted as before. Our present programs apply this method iteratively until no further clustering takes place.

The clustering process thus causes the formation of additional new high level sets. These are shown to the user who (if he wishes) can give them a name (external representation) which will be used in future, thus raising the level of the man-machine dialogue. These also save space by permitting factoring of the data as in the following example. The logical clustering is performed by collecting the entities into sets, then storing common Facts at the set level (Fig. 3).

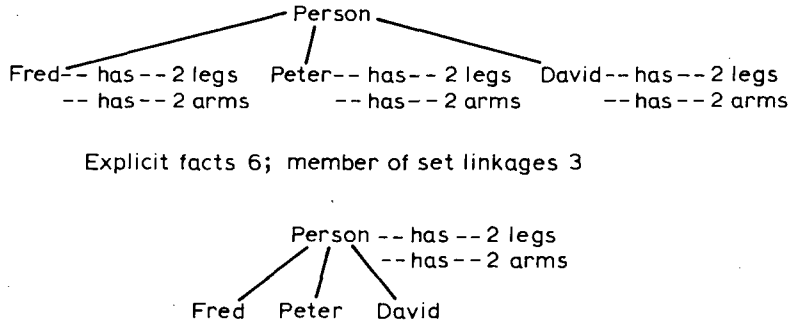

FIG. 3. Space saving by clustering

In general, given m entities each of which take part in n common Facts (i.e., which all have the same Relation and Object fields), this clustering process replaces the $n \times m$ explicit Facts by n Facts stored at the set level. The information thus stored is

implicitly available with all the m original individual set members by simple deduction (carried out automatically by the Inferential Fetch Operation) and the number of explicitly stored Facts is reduced by $n \times m-n$.

Clustering is also the basis of a simple method of Hypothesis formation. On finding a pair of entities which have a significant number of common properties and insignificant number of different properties the system will hypothesise that the entities are identical. The system converses with the user to determine if the hypothesis is correct. This has already been implemented. The system can be prevented from reconsidering rejected hypotheses by adding explicit 'denial of equivalence' Facts to the database. The 'significant' and 'insignificant' levels are adjustable parameters of the program.

5. GENERIC ASSOCIATIVE MEMORY

The Generic Associative Memory is a concept which developed during this project. Early versions of the Fact Machine had an abstract machine with a simple associative mechanism operating on a Fact store with a quadruple structure. In such a machine a 'Simple Fetch' would select all the Facts from the Fact store matching the entity or 'don't care' symbols placed in the Associative Memory Access Register. Such an abstract machine can be programmed to effect the generic retrieval (as described in Section 3) but requires as many associative cycles as there are nodes in the 'broom'. A Generic Associative Memory is one which requires only two cycles to form a complete 'broom' — one for the 'downwards', the other for the 'upwards' closure for any principal node in a single operation.

This high-level operation has been implemented directly in the most recent versions of our system. The reasons for first, defining this as an operation, and second, for implementing it by special mechanisms, are:

1. The operation is exceedingly useful in providing user-friendly retrieval. It is virtually essential if users are not to be disrupted by dynamic restructuring of the internal data model (e.g., after clustering).
2. By implementing it specially we can provide the hardware/software required for performance.

It was observed during the experimental work that the generic 'broom' network could be implemented on a hard-wired network (Fig. 4).

Applying a positive voltage to a particular node activates all the others in the powerset, while a negative voltage could similarly indicate the corresponding 'upwards closure'. This is an exceedingly rapid operation, depending only on the speed of propagation of the wired circuit but the mechanism seemed difficult to produce except as a permanently-wired network (unless perhaps by an automatic backplane-wiring machine!). We realised however that the network was analogous to a telephone system and thus we replaced the permanently-wired network by an equivalent one constructed as a crossbar exchange. Now the setting of the paths occurs by setting of various switches in the exchange (Fig. 5).

As in a telephone exchange however the number of switches in a crossbar scheme increases as N^2 where N is the number of external terminals (which now represent the nodes off the network), yet the actual number of switches ever 'closed' in an actual database is much less than N^2. We have overcome this by

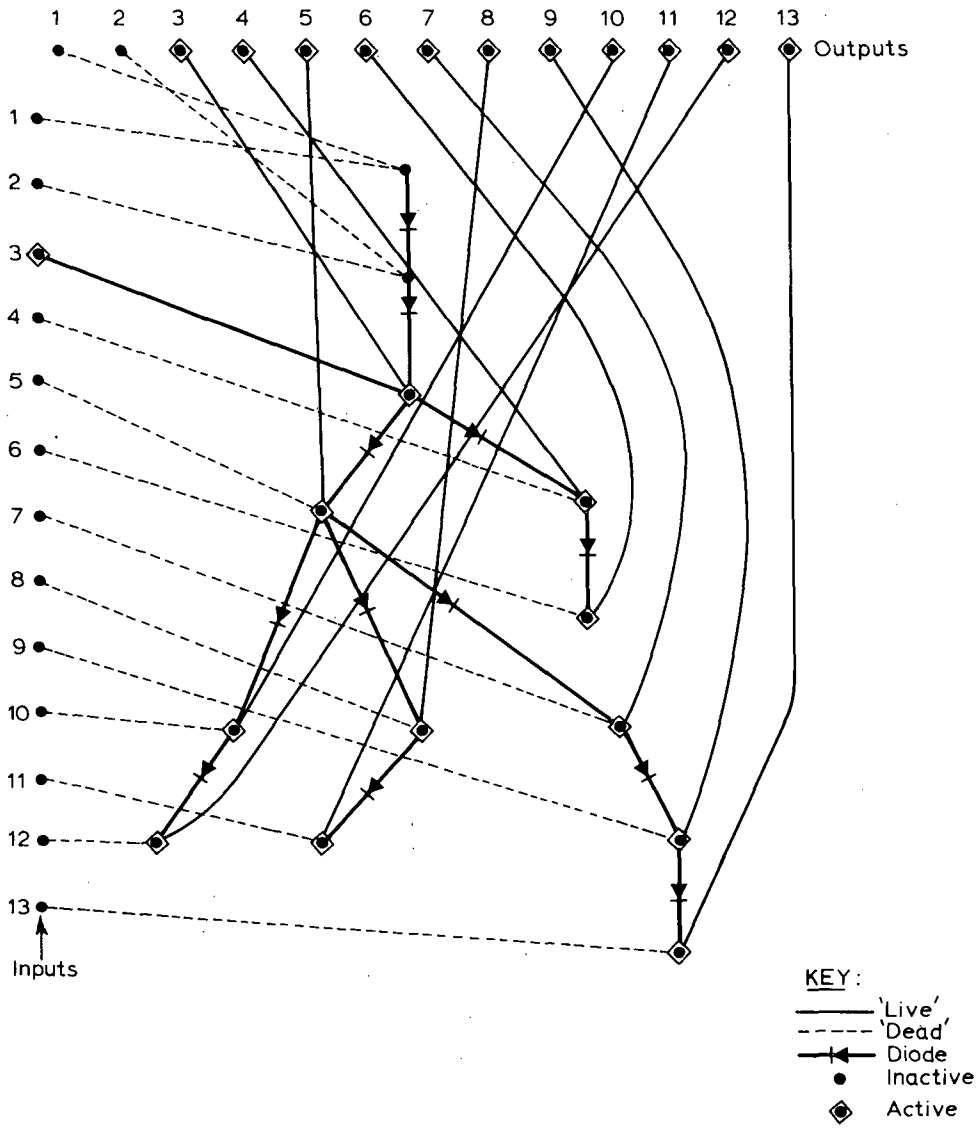

FIG. 4. 'Downward' closure of 'person'.' A 'Wired' network

1. Replacing the large crossbar by a number of small crossbar memories acting as though they were portions of the larger matrix.
2. Providing a mechanism to deal with the inevitable overflow of the smaller sub-networks by allowing the overflow from one sub-network to connect to another, and allocating these as necessary as the database grows.
3. 'Physical clustering' of the allocation of entities to network terminals so as to minimise the number of inter-block connections. It should be noted that this 'physical clustering' is distinct from the 'logical clustering' of entities into sets described in Section 4.

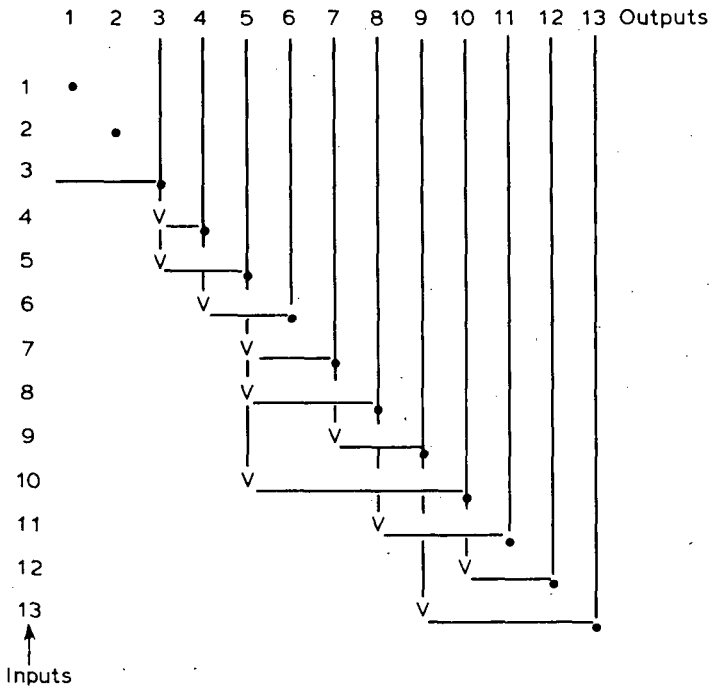

Fig. 5. Crossbar representation of network. V (diode) = switched connection; ● (wired) = permanent connection

We have now produced an outline design for an electronic memory capable of being implemented on an LSI chip capable of being 'set' like an ordinary RAM, but which exhibits the Generic Associative behaviour (Fig. 6).

We have also been investigating the performance of systems incorporating the new element by software simulation. To our surprise this has yielded by far the fastest systems we have implemented to date, partly because this is an extremely compact representation, partly because the allocation of entities to 'blocks' of the network results in a highly desirable 'clustering' of entities which are closely associated.

6. A DATA STRUCTURE FOR IMPLEMENTATION BY SOFTWARE

We shall now describe the data structure used by our software implementation. As is usual with database systems, data is presented to the system — and is returned from the system — in the form of variable-length character strings. These are converted by a scanner mechanism which uses a large backing-store file as a hash-table, and which returns a fixed-length identifier. In the case of a character string already present in the system, the identifier is of the form <net> <line>. <net> is a datablock representing the hardware net switch matrix; <line> is the row of the matrix allocated to this symbol. The required initial datablock can thus be accessed directly and fetched into mainstore.

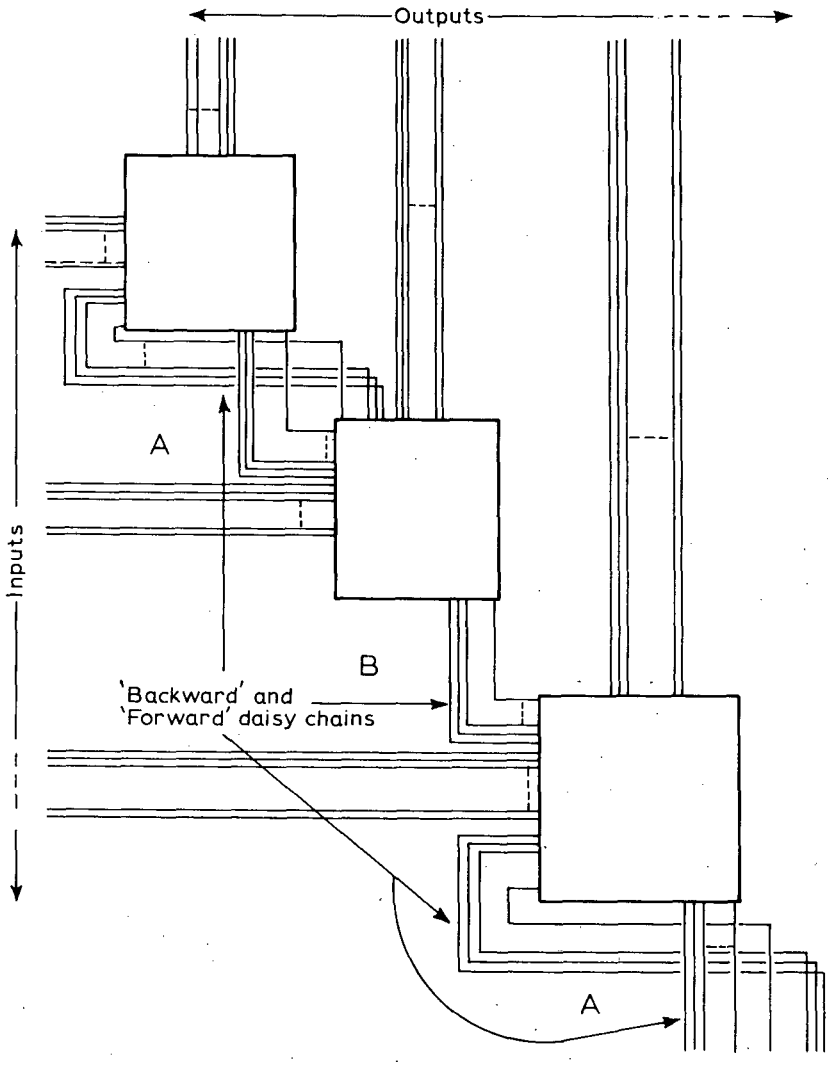

FIG. 6. Electronic memory exhibiting Generic Associative behaviour.
Note: Only a single set of (bi-directional) links is actually required (B) to achieve the effect as at (A), i.e., all links are bi-directional

The structure of this datablock is given in Figure 7. It consists of an $n*n$ square bit matrix, associated pointers, and 'buckets' containing data elements which can be chained to others making a cellular list. The bit matrix has one row for each entity which has been assigned to the net; the pointers refer to a descriptor of the additional information about the entity. Each descriptor contains:

nc Number of continuations (other nets) for this entity.
ns Number of Facts in which this entity is in the Subject position.
nr Number of Facts in which it is in the Relation position.
no Number of Facts in which it is in the Object position.

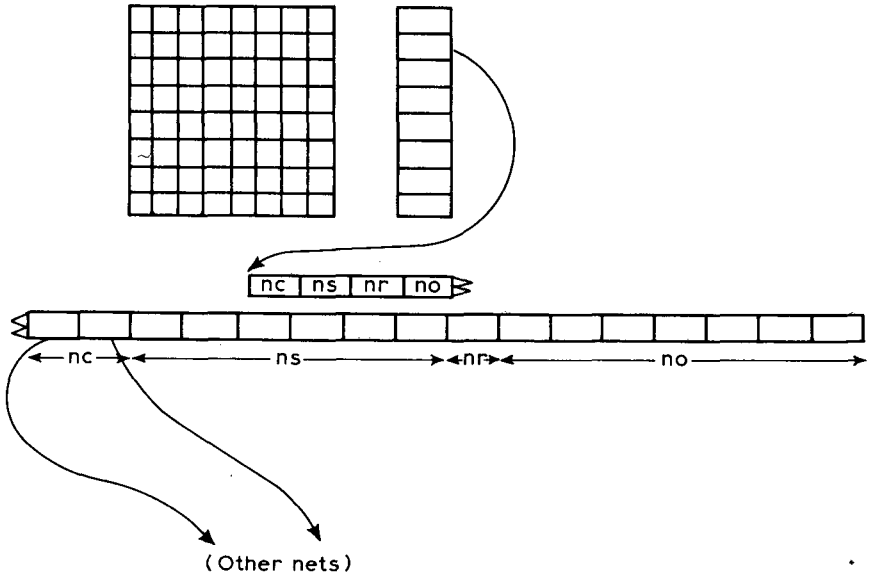

Fig. 7. Structure of datablock

These are then followed by contiguous groups of data elements representing Fact Numbers or entities.

The structure is designed to store each piece of information as closely as possible to the other information with which it is associated. When a new entity comes into the system, an identifier is allocated to it (which is also of the form <net> <line>) and the <net> is chosen to be one which:

1. has sufficient space;
2. contains associated data.

If no associated data are already present the new entity is allocated to a new or relatively empty block. When a new entity is associated with other information, but lack of space prevents it being allocated to the existing net, a connection or link must be made (using the pointers) from the existing net to a new net. A net is thus used as a small contiguous bucket to which entities can be assigned irrespective of their type. Although it may at first sight seem an untidy structure, it does in fact provide a good model for the clusters of associated items of information met in actual usage. It is particularly useful where we have to map the resultant data structure onto a backing store device.

7. PERFORMANCE: SCALING RULES FOR A 'NET'-BASED SYSTEM

Perhaps the major question about the semantic net approach is whether implementation techniques will permit scaling-up the systems to deal with databases of practical size. We shall now derive a simple scaling-rule for our system, and in a later section we shall compare its predictions with experimental measurements.

Assuming square nets of order n, what are the costs of accessing a set of size S?

As indicated above, initial conversion from character string to <net><line> identifier is carried out by a symbol-table mechanism which uses a hash-look-up. This requires approximately one access to the backing-store.

Accessing a large set — or recursive set — will require approximately $S/(n—1)$ backing-store accesses. (We shall consider the effect of a large-page virtual storage system later.)

With a suitable hash-function and a sufficient size of hash-table, the number of accesses to the backing store can be made independent of the overall size of the database. As the nets accessed in the retrieval are predominantly those which contain set members, and these have a high concentration of set members to a first approximation retrieval time is:

1. Proportional to set cardinality.
2. Independent of the overall size of the database (i.e., retrieval costs are not affected by the presence of data disjoint to the present operation).

In a practical system the costs may be further reduced by the use of techniques which further cluster the relevant information.

In our PDP11 system we adopted a mainstore 'cache' to hold the most frequently accessed nets ('second chance' replacement). This has proved highly beneficial when a series of operations is carried out on approximately the same data, and the working set of nets actually accessed is smaller than the mainstore cache.

For our larger ICL2980-based implementation we have used a placement strategy so that a series of nets containing information about the same large set are allocated contiguous physical storage within the same pagebuffer in virtual store. This reduces fragmentation and hence reduces the backing-store accesses required.

Given a pagesize p bytes, and a netsize of nv bytes, we have:

$$\text{No. of accesses} = \lceil \lceil S/(n—1) \rceil/(p/nv) \rceil$$

With some typical values ($n=32$, $p=4096$, $nv=2048$) this formula indicates that approximately 100 backingstore accesses are required for a set with some 6400 members.

Experimental results are summarised in Tables 1 and 3 (PDP11), 2 and 4 (2900-EMAS). They show that the scaling rule given is applicable, at least for these implementations. (Unfortunately in the EMAS/2980 experiments backing-store accesses must be indirectly inferred from system supplied information, and results are dependent on the general load on the system.)

Table 1. System 11/34 UNIX. Retrieval of 32 records from a disjoint database

Database size (no. of records)	*Actual response time* (s)	*Predicted disk accesses*	*Actual disk accesses*
32	2.1	33	33
82	2.0	33	33
132	2.1	33	33
1032	2.1	33	33
1532	2.2	33	33

Table 2. System 2980 EMAS. Retrieval of 32 records from a disjoint database

Database size (no. of records)	*Actual response time* (s)	*Predicted disk accesses*	*Actual disk accesses*
32	1.2	33	11
42	1.2	33	11
132	1.2	33	11
532	1.2	33	11
1032	1.2	33	11
3032	1.2	33	11

Table 3. System 11/34 UNIX. Performing a closure of increasing cardinality

Retrieval size (no. of members)	*Elapsed time* (s)
10	1.1
100	8.0
1000	24.2

Table 4. System 2980 EMAS. Performing a closure of increasing cardinality

Retrieval size (no. of members)	*Elapsed time* (s)
10	0.5
100	1.0
500	2.2
1000	5.5
3000	15.4

8. IMPLEMENTATION HISTORY

Implementation of the system has proceeded through several stages. The initial versions used a simple simulation of a content addressable Fact store — each access required a complete search of the store and all data were kept as an array in mainstore. We soon progressed to version 2 — which gave moderate performance on a larger volume of data located on the backing store on a conventional computer. Reasonably rapid access to the data was achieved by normal data processing techniques — we kept both direct and inverted files. Look-up used 'hashing' to achieve rapid access to the backing store, and an instore cache system was used for the most frequently used data. However, for the reasons already indicated generic retrieval always required multiple accesses to the backing store. Our third version of the system is actually a modified simulation of the network. As the hard-wired network would have much greater performance than is required (at present) for typical applications the simulation may well prove adequate. This version has cut the time required for one of our 'benchmarks' (compiling a small database) from 7 min to only 12 s (on a PDP11/34). We envisage a scaled-up version as shown in Figure 8.

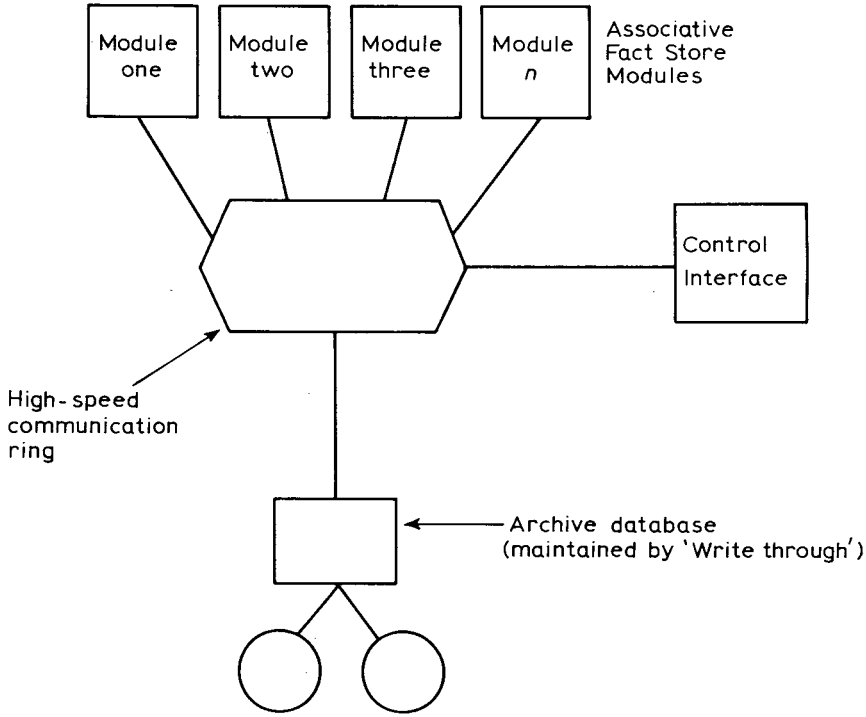

FIG. 8. Large system outline

9. COMPARISON WITH OTHER SYSTEMS

Early systems based on entities were reviewed by Winograd (1972). We saw the model as a development of Senko's DIAMII (1975), and an extension of our own work on special-purpose hardware for database systems (McGregor *et al.*, 1976). Chen (1976), Smith and Smith (1977) have worked on Entity sets, and generic information. The work of Lea (1975) on associative hardware was also influential. After our initial developments we discovered the work by Sharman and Winterbottom (1979), and were interested by the comparison between our basic associative Fact store and the programming language LEAP (Feldman and Rovner, 1969). Recent work by Deliyani and Kowalski (1979) on logic and semantic networks is somewhat similar to our own in its use of deduction but without the more general abilities of the Fact machine.

Semantic nets are a recognised method of representing knowledge in computer systems though they have only recently been applied to database systems (Rousopolous and Mylopolous, 1975; Sharman and Winterbottom, 1979). They have been found especially useful in the field of artificial intelligence where there is a wealth of literature describing various forms which they can take. A number of related systems have been developed by various workers — 'networks' (Woods, 1975), 'frames' (Minsky, 1975), 'generalisation hierarchies' Winograd (1975). There seems little doubt that nets are a powerful technique for the representation of information, though there is much debate on which representation is 'best' for a particular application.

In general, earlier implementations of semantic nets were unsuitable for use in large scale databases. Usually the data were held in mainstore, and explicit pointers were used to indicate links between nodes on the net. Later systems, in particular the implementation of Sharman and Winterbottom, placed semantic nets on the backing store. Here however, the links between nodes were still explicit (address) pointers.

Size comparisons with more conventional database systems are complicated because we are not comparing like with like. We suggest that there are three recognisably different measures:

1. Actual storage size — the actual stored volume.
2. Storage-equivalent size — the volumes of systems fed the same set of input data.
3. Query-equivalent size — the volumes of systems capable of responding correctly to the same set of queries.

We normally experiment with databases of about 20000 Facts, and the system handles these volumes easily, giving acceptable response times. The system is loaded from some 2000 150-byte records. This volume of information is storage-equivalent to a conventional file of records of about 0.3 megabyte. However this size is in no sense an upper limit, present work is aimed at producing a demonstration system with an order of magnitude increase in size. The actual stored size is approximately 0.3 megabyte, while the query-equivalent size must correspond to a conventional database of some tens of megabytes, as each term is multiply-indexed.

10. CONCLUSION

We believe that the use of generic information to provide a self-adaptive user interface, through which users and other computer systems can immediately (and without prior rigid standardisation on a specific thesaurus of agreed terms) achieve effective communication, is a significant and highly-valuable ability. This will be particularly useful in the context of distributed database systems. The ability of the system to utilise entity external character string denotations in a variety of human natural languages is a useful side-effect.

The system treats information in an orthogonal manner. There is no difference in storage, or access method between schema and data, nor is there a separation of catalogue and data dictionary from data files as there is in most other database systems. Even operational information and 'programs' which form part of the system are held in the same standard way as all other information. The data model is capable of representing knowledge to any desired level of detail, in a form independent of the character strings or other denotations originally employed to insert it into the system. Further, the 'secret' nature of the internal knowledge structure, combined with the automatic intervention of broadening, deduction, transitive and reflexive relations and special purpose procedures (Computed Facts and Computed Relations), gives the system a unique degree of data independence. The internal data structure can be completely re-organized without any external user or application being in any way affected by the change.

We have shown that the generic network can be represented by a hard-wired network, but more significantly, that this in turn can be implemented by means of a sparse matrix of storage elements made up from regular sub-matrices of moderate

size. In this way we believe a compact high-speed associative system can be constructed using LSI components.

ACKNOWLEDGEMENTS

We should like to thank our colleagues in the Department of Computer Science for their advice and helpful discussions. The work is supported in part by S.R.C. Grant GR/A/3682.7.

REFERENCES

Bagshaw, S. and McGregor, D. R. (1980) Disc cache for minicomputer systems. *Systems International 2*, 41–43.

Chen, P. P. S. (1976) The entity-relationship model. *ACM Transactions on Database Systems 1*, 9–36.

Codd, E. F. (1970) A relational model of data for large shared data banks. *Communications of the ACM 13*, 377–387.

Deliyani, A. and Kowalski, R. A. (1979) Logic and semantic networks. *Communications of the ACM 22*, 184–193.

Fahlman, S. E. (1979) NETL: A system for representing and using real-world knowledge. Boston: M.I.T.

Feldman, J. A. and Rovner, P. D. (1969) An Algol based associative language. *Communications of the ACM 12*, 439–449.

Lea, R. M. (1980) Associative processing of non-numerical information. Proc. Summer Institute NATO ASI series c-32 171–215.

Lea, R. M. and Schuegraf, E. J. (1980) An associative file store using fragments for run-time indexing and compression. *Information Retrieval Research* (R. N. Oddy, S. E. Robertson, C. J. van Rijsbergen, P. W. Williams, eds.) pp. 280–295. London: Butterworths.

McGregor, D. R. and Malone, J. R. (1980) The fact database: a system based on inferential methods. *Information Retrieval Research* (R. N. Oddy, S. E. Robertson, C. J. van Rijsbergen, P. W. Williams, eds.) pp. 203–217. London: Butterworths.

McGregor, D. R., Thomson, G. T. and Dawson, W. D. (1976) High performance hardware for database systems. *Proceedings of the Second Conference on Very Large Databases* (G. M. Nijssen, ed.) Preprints 103–116. Amsterdam: North Holland.

Minsky, M. (1975) A framework for representing knowledge. *The Psychology of Computer Vision* (P. H. Winston, ed.) McGraw-Hill.

Rousopolous, N. and Mylopolous, J. (1975) Using semantic networks for database management. *Proceedings of the First Conference on Very Large Databases* (D. Kerr, ed.) pp. 144–172. Amsterdam: North Holland.

Senko, M. E. (1975) Specification of stored data structures and desired results in DIAM II with FORAL. *Proceedings of the First Conference on Very Large Databases (1)* (D. Kerr, ed.) pp. 557–571. Amsterdam: North Holland.

Sharman, G. O. H. and Winterbottom, N. (1979) *NDB: Non-programmer database facility.* Hursley (IBM Technical Report TR 12.179).

Smith, J. M. and Smith, D. C. P. (1977) Database abstraction: aggregation and generalisation. *ACM Transactions on Database Systems 2*, 105–133.

Winograd, T. (1972) *Understanding Natural Language.* Edinburgh University Press.

Winograd, T. (1975) Frame representation and the declarative/procedural controversy. *Representation and Understanding.* (D. G. Bobrow and A. Collins, eds.) New York: Academic Press.

Woods, W. (1975) Foundations for semantic networks. *Representation and Understanding.* (D. G. Bobrow and A. Collins, eds.) New York: Academic Press.