SRC TR 87-177

Expert Database Systems: Efficient Support for Engineering Environments

by

T. Sellis, N. Roussopoulos, L. Mark and C. Faloutsos

EXPERT DATABASE SYSTEMS: EFFICIENT SUPPORT FOR ENGINEERING ENVIRONMENTS

T. Sellis, N. Roussopoulos, L. Mark and C. Faloutsos

Department of Computer Science
and Systems Research Center
University of Maryland
College Park, MD 20742

Abstract

Manufacturing and Engineering processes use both large scale data and knowledge bases, and the use of expert systems in such environments has become a necessity. Expert Database Systems have evolved from conventional database systems to meet the requirements of current Artificial Intelligence applications. However, future Expert Database Systems will contain knowledge bases of significant size which makes main memory insufficient and the use of a database system a necessity. We propose an effective way of building High Performance Expert Database Systems to support manufacturing and engineering environments. These systems are based on *Incremental Computation Models*; such models utilize results of previous computations by merging them with newly derived results of computations on small increments representing changes in the database. Our system will be able to support very large knowledge bases by utilizing novel structures and access methods and by using a very sophisticated inference engine based on incremental computation models.

This research was sponsored partially by the National Science Foundation under the grant CDR-85-00108 and by UMIACS.

1. Introduction

Traditionally Database Management Systems (DBMS) have been used in business applications to efficiently store and organize large amounts of data. The main thrust of database research has focused on designing data structures and algorithms so that operations, common in this environment, can be processed efficiently.

Recently, there has been considerable interest in providing a framework for information sharing and exchange in engineering environments. A lot of attention has been given to consolidate the engineering support environments that have been and are being developed to support design automation, manufacturing, resource management, planning, etc. Along this direction, there has already been some work in extending existing Database Management Systems to accommodate engineering applications. In particular, relational DBMS's have been used in support of Computer Aided Design (CAD) [16,23], Computer Integrated Manufacturing (CIM) [19,20], and Artificial Intelligence and Expert Systems [17,18]. The main difference between the business applications and the ones mentioned above lies in the kind of information that the two types of applications are using. Business applications are mainly concerned with large volumes of *structured data*, while Artificial Intelligence or Engineering Applications usually involve a sophisticated *control mechanism* that handles structured and unstructured data. Therefore, a system of the second type should be able to support the storing and handling of control information in addition to data.

Using a data manager with full capabilities offers the advantages of better data organization, simple user interface, integrity of data in multi–user environments and recovering from hardware or software crashes. Given these advantages, there have been various attempts to build systems that support non–traditional database applications over large volumes of data. In general, there are three different approaches that can be taken

- One can enhance a specific application system with a *specialized* data manager
- One can *interface* a specific application to a general purpose DBMS
- Finally, one can *extend* a general purpose data manager by enhancing it with more sophisticated capabilities (e.g. inference, triggers, etc).

The first approach suffers from two major disadvantages. First, considerable effort must be put into designing and building several modules that DBMS's already include. Second, such specialized data managers are very narrow, in the sense that they cannot be easily modified to support applications other than the ones they were originally written for. In the second approach the DBMS acts as a server to the application program by supplying on demand the data that the latter requires. However, the major disadvantage of this approach lies in the difficulty to define exactly where the two systems must be interfaced. [2] provides a good criticism of this approach.

Because of the above mentioned difficulties, data managers with extended capabilities have been proposed. The work of [29,44,50] in semantic data models, of [5,7] in the design of systems based on the object oriented programming paradigm and of [47] in extending INGRES to support expert system applications, are representative of this approach. The basic idea is to come up with a simple system that gives to the user the capability to *build on top* of a basic set of functions whatever constructs are required by specific applications. Moreover, it has been assumed that minimal extensions to the relational model should be attempted. An example of such efforts have been *Engineering Databases* [1,24]. The assumption here is that complex objects can be supported by normalizing them into relational structures connected via the concept of **surrogates**.

Another example of a similar approach is the work on *Deductive Databases* [12]. The direction there was to provide basic support for expert systems applications. In a deductive database system both deductive aspects of the world (*rules*) and asserted information (*facts*) are stored in the same system. The framework represented by logic programming [21] and typified by the programming language PROLOG, is used as a common example. However, because of well known problems with the query processing algorithm (tuple–at–a–time), the artificial control strategies used (cut,fail) and the lack of organizational principles for large knowledge bases in PROLOG, various researchers have been engaged in designing extensions of DBMS's instead of trying to interface PROLOG or a general inference engine to a data manager. In [6,49,51] several designs for database systems enhanced with inference capabilities are proposed, each being a specific implementation of the above model of rules and facts.

Although the above mentioned proposals suggest basic models for engineering complex objects and deductive systems, they do not explore in detail all expert systems requirements that need be incorporated into relational database systems. Engineering and deductive databases are severely handicapped by their performance. Little work has been done to improve access methods for storing large rule bases, redundant structures, intelligent query processing and efficient support for procedural and heuristic knowledge. In this paper we discuss such issues and suggest solutions.

Section 2 briefly presents what are the basic expert systems requirements. In Section 3 we show how such requirements can be handled by minimally extended relational database systems. Section 4 discusses the various performance and optimization issues and suggests various solutions. Then in Section 5 we report on the implementation status of this effort. Finally, we conclude with Section 6 by summarizing our ideas and pointing out areas of future research.

2. Expert Systems Requirements

The ultimate goal of Expert Database Systems (EDS's) is to provide an efficient alternative testbed for the implementation of expert systems. The main reason for following such an approach is the constantly increasing size of the information that needs be managed by current and future expert systems. Systems like INTERNIST-1 [28], R1 [30], PROSPECTOR [8] and DENDRAL [22] manipulate (not necessarily in an efficient way) information in the order of thousands of facts and rules. Frederick Hayes-Roth very recently mentioned in [14] that he expects knowledge bases to have in the order of 100,000 rules by 1990, instead of 3–10,000 which is the current figure. Organizing the information using the well defined principles of database systems would clearly be an advantage. However, one must first look in a systematic way at all the basic expert systems' requirements in order to get a good understanding of the kind of extensions that need be made to current database systems. This section provides such a discussion.

We shall first introduce a simple example and use this as a medium for the subsequent discussion. The system is shown in Figure 1.

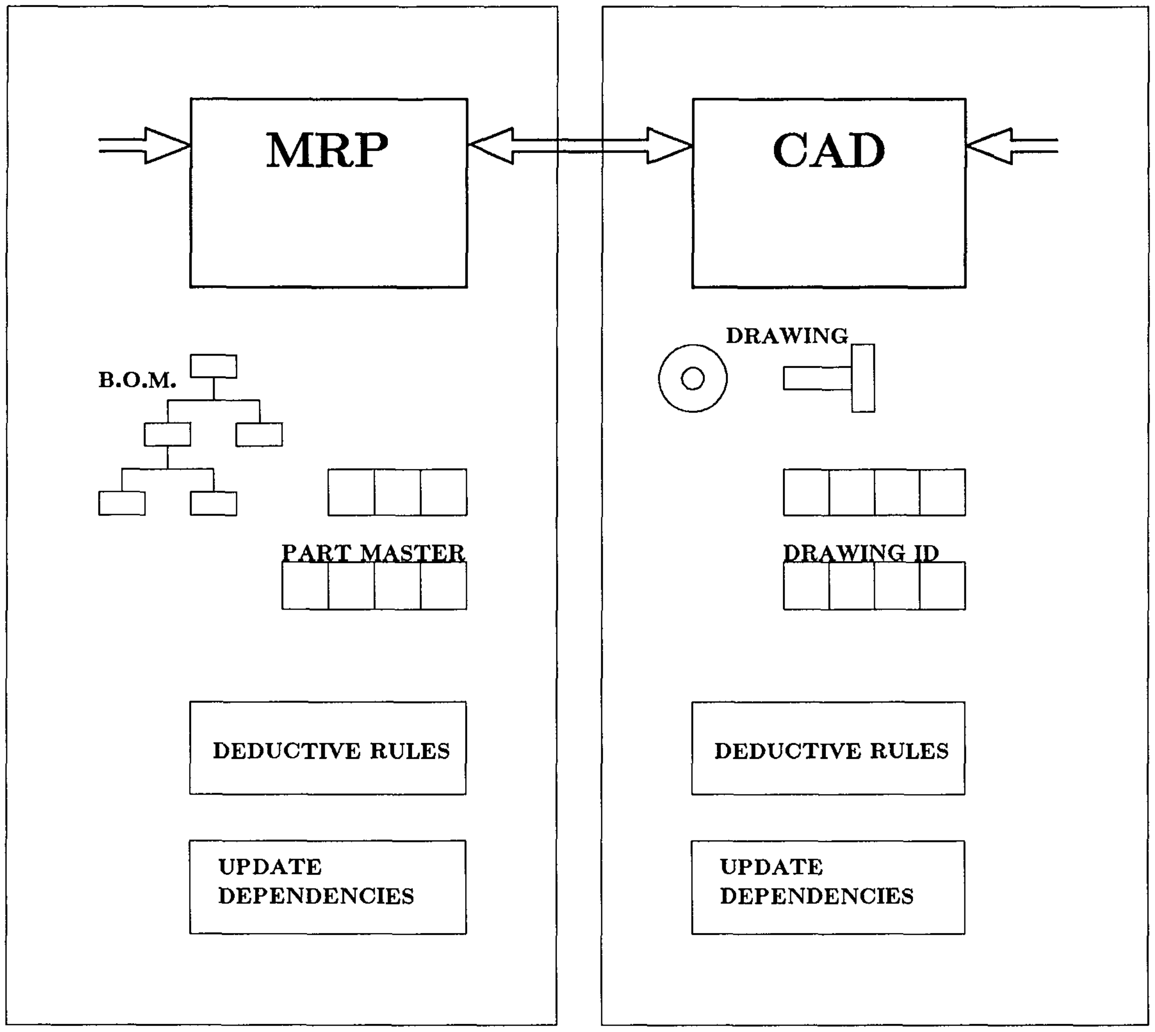

Figure 1: An Example EIS

The example illustrates a simplified model of an information system for Manufacturing and Resource Planning (MRP) integrated with an information system for Computer Aided Design (CAD). The MRP system contains a variety of information, in particular the Bill of Material (B.O.M.) describing part explosion and the Part Master record containing the individual part descriptions and versions. The CAD system contains designs and drawings in various versions. As we shall see later, there is a number of situations where deductive rules are needed for complex data and information retrieval from the two systems. There is a considerable amount of overlap and redundancy between the data stored and used by the two systems. The update dependencies of such redundant data systems require explicit representations to be used by the redundancy controller in maintaining consistency, in supporting data exchange and change propagation between the two systems.

By definition, an Expert System (ES) is a computer system which attempts to behave like a human expert in some limited application domain [4]. Looking closer at the architecture of most ES's, we can identify as its major components the **Knowledge Base** (KB), the **Inference Engine** (or "control engine") and the **Learning Module**. In the following we analyze in more detail the structure of those components.

2.1. The Knowledge Base

The knowledge base contains all the information needed to carry out the tasks that the ES will be given. It consists of **Specific Knowledge** (or data) and **General Knowledge**. Specific knowledge corresponds to data that has been acquired, like for example measurements or statistical data, and it is usually stored in the system in a very structured way. The main difference between an ES and another piece of software is that data is stored in a declarative way and is not hidden somewhere in a piece of code. General knowledge can be thought of as the intentional part of the knowledge base. It contains general principles, rules and problem–solving heuristics and corresponds to the human expert's accumulated empirical and/or technical knowledge.

We will now briefly look at the properties that a knowledge base usually has. First, in the same way a database system uses a specific model to describe the data, several techniques exist to represent the general knowledge required for an expert system (*Knowledge Representation Schemes*). Second, knowledge can be static or dynamic. For example, data stored in the system can be retrieved while on the other hand general knowledge can be used to derive new facts. Finally, a significant component of the knowledge base, called **Meta–Knowledge**, is used to describe the scope, precision, reliability and other properties of the stored knowledge.

2.1.1. Representation schemes

There are several knowledge representation techniques (see [29] for a thorough discussion). However, the following four are the most commonly used ones

1. First–Order Logic: In this scheme, specific knowledge is represented through first–order predicates and general knowledge is expressed through axioms. PROLOG is a typical example of a system that represents its knowledge base using a subset of first–order logic (Horn clauses). Although this scheme offers well defined semantics, simple notation, representational uniformity and the simplicity of being only declarative, it suffers from two major drawbacks, namely the lack of good organizational principles for large knowledge bases and the absence of any mechanisms to represent procedural and heuristic knowledge.

2. Semantic Networks: Networks are also a very natural way to organize knowledge. In addition they provide efficient organization and offer good heuristic mechanisms for manipulating the knowledge. Their major drawback stems from the lack of formal semantics and the lack of mechanisms for storing procedures.

3. Frames: This scheme offers a very general and powerful representation model. Frames provide good organizational principles and they are quite general and powerful. Both declarative and procedural knowledge can be represented through types, values and procedures attached to slots of frames. Perhaps their major problem is efficient implementation.

4. Production Rules: This is probably the most popular form of knowledge representation in expert systems. The major advantage of production rules is that they offer a very general framework under which most of the expert systems' knowledge bases can be represented. They share the advantages and disadvantages mentioned above for frames.

2.1.2. Dynamic vs static knowledge

One possible classification of the knowledge components can be based on the way they are used during the inference process. There is a static (or "passive") component, which corresponds to data stored explicitly in the KB. And, there is a dynamic (or "active") component which can produce knowledge. Rules or attached procedures in the case of frames are examples of dynamic knowledge components. In some cases knowledge is retrieved from the system and combined in various ways to produce new knowledge, like for example PROLOG does when applying a rule on ground facts. In other cases, specific new knowledge can be constructed and *actually* stored or removed from the KB. This is the case when procedures are used to modify the KB with the addition of new or deletion of old data.

2.1.3. Describing the knowledge base

Another component of the knowledge base is **Meta–Knowledge**. This is used to describe the knowledge base itself. Examples of meta–knowledge include

- Description of the semantics of data (e.g. types, constraints)
- Importance and precision of knowledge
- Heuristics to improve the performance of the inference engine

In most systems meta–knowledge is expressed in a procedural way and mainly represents heuristics that cannot be otherwise incorporated into the system.

2.2. Inference Engine

The inference engine of an ES clearly depends on the knowledge representation scheme used. For example, a production–rule based system may support *forward chaining.* That is, given a situation X and a goal G, find a series of rules which if applied starting from X will lead to G. Another approach however may be *backward chaining* where one really tries to "prove" G by establishing X through further analysis. Finally, *bidirectional chaining* is a combination of the above.

Reasoning in semantic networks corresponds to network traversals and matching. Frames are handled similarly with the addition of procedure fire–ups while rules in first–order logic based systems are interpreted using a backward chaining procedure. In general, some combination of the above methods can be used for the inference engine of an ES. Because rules seem to be the most widely used representation scheme, the discussion to follow in section 3, is restricted to backward/forward chaining only.

2.3. Learning Module

The learning module is used to enrich the knowledge base of the system using past experience. Although several research issues have been studied in the past or are under investigation in the general area of AI learning, we have not yet seen any of these ideas specifically applied in the area of expert systems. There are a few ideas on learning by experience that can be explored

- An ES can learn to be fast in certain cases. In that sense, the ES simply optimizes its rule processing mechanism to work efficiently in some pre–determined cases.
- An ES can enhance its knowledge by storing previously made derivations. For example, in a first–order logic based system, the system may choose to store the results of some previous computations. If the same requests are issued frequently, the response time of the system will be greatly improved.
- An ES can gather data and statistics from user sessions to improve performance based on heuristic procedures for cost estimation, etc.
- An ES can observe regularities in the given data, and perform induction and generalizations, that is, suggest new rules that are hidden behind the regularities.

Of course these are just a few of the issues that may arise in this area. However, we feel that the learning by experience module is an important component of an ES.

Before turning our attention to the impact that intelligent database systems may have to the development of expert systems, we provide a classification of the basic operations performed by an ES. These are

- Specific or general knowledge retrieval, either as end result to the user or as an intermediate step in the inference engine operation.
- Inference, through either of the methods mentioned above.
- Reasoning. The ES must be able to reason about its work. This is especially useful for the user to understand how the system reached a specific result.
- Maintenance of knowledge base. This corresponds to the result of modifications made to either general or specific knowledge.

Given these expert system characteristics we move now to discuss how advanced database management systems can play a key role in the development of large scale expert systems.

3. Mapping Expert Systems Components to DBMS Features

In this section we suggest schemes for mapping expert systems characteristics to relational database systems features. Although the discussion to follow presents ideas mainly applicable to production–rule based systems, semantic networks and frames could also be supported. The basic features discussed are data, rules, meta–knowledge and the inference mechanism. Throughout the discussion we will illustrate our proposals through the MRP/CAD system of section 2.

Specific knowledge can be clearly stored using relations. For example, the realization of a predicate `bom(part,sub-part)` is done using a relation `BOM` (we will use all–capital names to distinguish DBMS entities from ES ones) with two fields `PART` and `SUBPART` and specific domains for these two fields.

Rules can be also mapped to database entities. We will discuss two types of rules here. First, consider simple inference rules such as those used in deductive databases. For example, a rule may be used to define the `contains` predicate as follows

```
contains(cover,part) ← bom(cover,X) ∧ bom(X,part)
```

The above rule can be expressed using relational views as follows (we use SQL to describe the view)

```
CREATE VIEW CONTAINS (COVER,PART)
    AS SELECT FIRST.PART, SECOND.SUBPART
       FROM   BOM FIRST, BOM SECOND
       WHERE  FIRST.SUBPART = SECOND.PART
```

The semantics are exactly the same, that is, they both define the cover–part entity in the same way by examining the `bom` entity.

A different type of rules are general production rules of the form

```
IF  <CONDITION>  THEN  <ACTION>
```

where `ACTION` is a general operation such as an insertion or deletion, not just a retrieval from the KB as deductive rules assume. Hence, this kind of rules incorporate procedural semantics in ES's. Notice that this type of rules corresponds to the idea of triggers or alerters [3] in database systems. They can be used to generally model such activities as well as protection and security mechanisms.

This kind of rules can be modeled in a database system using the idea of *Update Dependencies*. The update dependency formalism was originally suggested in [25,26] to support a wide variety of applications, such as walk–through guidance control systems, cause–effect systems, statistical information gathering, knowledge acquisition, database integrity enforcement, database view updates, policy enforcement, and production control. An update dependency has the following format

$$\texttt{ON <OPERATION> PERFORM <ACTION}_1\texttt{,ACTION}_2\texttt{,...>}$$

where OPERATION is an operation on the database that results to a series of actions $\texttt{ACTION}_1$, $\texttt{ACTION}_2$, etc. Using update dependencies one can implement production rules by simply defining the OPERATION and ACTION parts. As an example, we consider again the MRP/CAD system. Suppose there are two databases, the **CAD** and the **MRP** database with the following format respectively

CAD drawing_id

part#	description	u.o.m.	status	revision_no

MRP part_master

part#	description	lead_time	cost	status	revision_no

The following example illustrates a rule that calls for implied operations on the **MRP** database when a new design is completed in the **CAD** database.

$$\begin{array}{ll}\texttt{ON} & [\ \texttt{complete}_{CAD}(\texttt{drawing\_id}_{CAD}(P,D,U,S,R)) \wedge \neg\texttt{drawing\_id}_{CAD}(P,\_,\_,S,R)\ ] \\ \texttt{PERFORM} & [\ \texttt{create}_{MRP}(\texttt{part\_master}_{MRP}(P,D,\_,\_,S,R)), \\ & \ \ \texttt{assert}(\texttt{drawing\_id}_{CAD}(P,D,U,S,R))\ ]\end{array}$$

$$\begin{array}{ll}\texttt{ON} & [\ \texttt{complete}_{CAD}(\texttt{drawing\_id}_{CAD}(P,D,U,S,R)) \wedge \texttt{drawing\_id}_{CAD}(P,\_,\_,S,R)\ ] \\ \texttt{PERFORM} & [\ \texttt{write}(\texttt{"Drawing already exists for"},P)\ ]\end{array}$$

The rule checks if a given drawing already exists in the database, and if not it adds it to the **CAD** database making the corresponding insertions in the **MRP** database as well.

Update dependencies are based on the logic programming formalism [21]. It can, therefore, be used to implement all known trigger mechanisms. However, because logic programming is found difficult by a large class of programmers, we have extended the formalism to include familiar control structures from ordinary programming languages such as *while*, *case*, and *repeat* statements [25].

A very significant part of a knowledge base lies in the **Meta–Knowledge**. This is an issue that has received very little attention in previous proposals for the implementation of expert database systems. Most of these proposals assume that meta–knowledge is hidden somehow in the various operations or the search algorithm. One exception may be in the design of POSTGRES [47] where priorities set for rules as well as extended data type definitions are actually themselves stored in the database. In this proposal we look for better and more complete means of describing meta–knowledge. Recent work in Self–Describing Databases and Meta–data Management [25,27] is applicable to the management of meta–knowledge. A self–describing database system maintains an Intension–Extension Dimension of data description that consists of four levels, each of which is the extension of the level above it and the intension for the level below it. The intension–extension dimension provides an active and integrated Data Dictionary System as part of a self–describing database system and is therefore ideal for meta–data management. The key issue in expert database systems will be to find an appropriate database structure for representing and retrieving rules and meta–rules. This issue is discussed in more detail in the next section. However, what is really significant here, is that the mechanism incorporating meta–knowledge exists and has been studied in detail.

Finally, we discuss briefly the implementation of an inference engine in an EDS. As illustrated above, both simple deductive rules and more general procedural rules can be supported; hence, the inference engine really maps to the query processing and data manipulation engines of the database system. Backward chaining is handled already through query modification for view processing [45]. For example, if one asks for the part containing `bolt1`, the database query will be

```
SELECT COVER
FROM   CONTAINS
WHERE  PART = "bolt1"
```

which is in turn changed using query modification to

```
SELECT FIRST.PART
FROM   BOM FIRST, BOM SECOND
WHERE  FIRST.SUBPART = SECOND.PART
AND    SECOND.SUBPART = "bolt1"
```

The query modification module needs to be extended to handle recursive view definitions [15] as well as multiple view definitions, but conceptually the mechanism exists already in database systems. Forward chaining can be implemented based on the triggering mechanism offered by update dependencies. Hence, appropriate design and implementation of an EDS can take advantage of all existing database technology to support expert systems. However, the major question arising in such an environment will be *performance*. This is really the focus of this paper and our ideas are discussed in the following section.

4. Optimization Issues

Expert Database Systems and Object Oriented Databases are severely handicapped by the performance of existing DBMS's. The database access and the real time response requirements are well beyond the capabilities of classical query optimization and buffer management techniques. Classical methods are based on static compilation of data processing access patterns. These patterns are very different from the very dynamic and specialized EDS ones necessary to support a general inference mechanism that an ES may employ. Adaptive access mechanisms for improving performance are necessary.

We propose the idea of *Incremental Learning* as a solution to the high cost of repetitive access patterns observed in Expert Systems. Incremental learning is a concept that enables a system to avoid unnecessary search by remembering some of the computation search it did before. It allows the generation of ever improving systems. The foundation of Incremental Learning is a new class of **Incremental Computation Models** [39].

In this section we discuss several optimization and implementation solutions based on Incremental Models. The issues that we will deal with include efficient processing of rules, specialized storage structures for incremental access methods, and ideas on indexing both large fact and rule bases.

4.1. Incremental Computation Models

Let C be a computation performed at time t_1 on input I_1. An Incremental Computation Model (ICM) performs C at time t_2 by merging the result obtained at time t_1 with the result obtained by performing C on the **differential** of the inputs ($dI=I_2-I_1$). An ICM can be employed in any distributive computation. The advantage of an ICM is clearly in performance because, at any time, the computation is only performed on the increments which are typically very small, and get emptied after each use. For example, if we need to produce the set of all cover–parts in our database, we merge the previously generated set with the recent ones.

The above incremental approach can be extended to account not only for identical computations but to those that are derivable from others. We can perform a derivable computation using the differential between the two computations $dC=C'-C$. Incrementally derivable computations are applicable to any monotonic computation. For example, extracting from the database only the parts including `bolt1`, can be done using the set `CONTAINS` produced before, i.e. the parts containing other parts, appropriately updated to reflect all the new containments introduced due to additions of new parts.

A wide class of computations performed by real–time systems can be accommodated by incremental computation models. Dramatic performance improvement is obtained when the increments are small. Expert Systems doing deductive search are the most representative ones because of the tiny increments between the huge number of one–fact–at–a–time access requests. ICM's also facilitate the dynamic establishment of learned search patterns pertinent to the application. Other classes of systems that will be greatly improved by ICM's include surveillance systems, control and command systems, air–traffic control systems, control of manufacturing processes, etc. Such systems receive data arriving in real time from simultaneous sensor and human observations and require rapid and intelligent assimilation of them with the help of factual information stored in the database. Most of these systems have very high input frequency that results in extremely small input increments. Therefore, ICM's can improve performance by at least an order of magnitude.

4.2. Rule processing and optimization

Currently, most expert systems and simple solutions to building EDS's, such as interfacing PROLOG with a DBMS, are characterized by *one-fact-at-a-time* access which drives the deductive search. Very often, access to a single fact requires a long database search. Consider the example of the previous section, where the join of the `BOM` relation with itself is performed to determine whether or not `CONTAINS(part1,bolt1)` is true. Imagine a search where a large list of `CONTAINS` pairs had to be checked using an ordinary system. For each pair, the join is to be constructed, accessed, and thrown away. Only to be repeated again, and again. The solution to this problem is to utilize existing database access paths to avoid such high searching costs. Since the join will be the most used operation in such a system, efficient support for its computation is needed.

We can exploit the idea of *Incremental Learning* for avoiding the high cost of repeating the same relational operators over and over. This can be done by saving a view storing what was learned in previous searches. In our parts example, a system could learn which of the part/sub-part pairs joined with other part/sub-part ones during the processing of the first query on `CONTAINS`, and *short cut* the join for the rest of the queries that may come in the future. This incremental learning avoids all the database search needed to reconstruct the join.

Incremental models and their learning capability generalize the utilization of common access paths, and permit a framework for the reuse of the optimization performed by the query optimizer. The optimization techniques will be transformed into inter-query incremental algorithms that will amortize their cost over a series of queries.

Another expensive computation is performed in Expert Systems because queries involve rules with more than one definition (more than one rule with the same "head" in deductive definitions). This expands the initial query to a set of queries to be processed by the system. In this case, support for multiple-query processing is needed. It is often advantageous to process a collection of queries differently than at the query-at-a-time manner. The reason is that queries may share data and therefore elimination of redundant page accesses may be possible. In [40,41] we study this problem and suggest several multiple-query processing algorithms. Performance improvements can be achieved at a very high degree depending on the extend that queries access common data. Since many expert systems applications have a lot of rules defining the same entity, support for multiple–query processing will be a significant part of the query optimizer.

4.3. Indexing of specific knowledge

As mentioned in the previous section, specific knowledge can be approximated by relations, each fact stored as a tuple. With few exceptions [48] expert systems assume that the knowledge base fits in main memory. When stored on disk, conventional indexing techniques can then be used to speed–up searching. We propose the use of *multi–attribute hashing* [33] to index large fact bases. The main advantage of multi–attribute hashing is speed, due to the reduced I/O activity it achieves. Using the Incremental Computation approach on top of multi–attribute hashing is rather easy. We expect that the combination will achieve excellent response times.

The basic idea behind multi–attribute hashing is clustering similar records in the same or consecutive buckets which results to better I/O performance. According to multi–attribute hashing, each record yields a bit–string $\vec{b}$ of size n ("record signature"), by applying a hashing function to each attribute value and combining (eg., concatenating) the binary representations of the hash values ("attribute signatures"). The binary value $(\vec{b})_2$ decides the bucket that the record is stored. For example, assume that the `BOM` relation contains the tuples

PART	SUBPART
`big-tank`	`fuel-tank`
`big-tank`	`large-compartment`
`large-compartment`	`wheels`
`fuel-tank`	`fuel-pipe`

and that the hashing functions $h_1()$, $h_2()$ for the first and second attribute respectively are

$$h_1(key) = h_2(key) = \begin{cases} 0 & \text{if } key \leq \mathtt{k} \\ 1 & \text{if } key > \mathtt{k} \end{cases}$$

Thus, the attribute signature of "`big-tank`" is h_1(`big–tank`) $= 0$, while the record signature of the tuple `(big-tank,fuel-tank)` is "00". In this example, the hash table consists of four buckets, with the following contents:

signature	PART	SUBPART
00	big-tank fuel-tank	fuel-tank fuel-pipe
01	big-tank	large-compartment
10		
11	large-compartment	wheels

Notice that, on partial match queries, we need only search a fraction of the hash table. For example, when searching for the sub-parts of "big-tank", we need only examine the first two buckets. Also, notice that the records of a bucket have identical signatures, and therefore similar attribute values. It is probable that such records will qualify for the same query. Thus, one bucket (= disk) access will retrieve many qualifying tuples. Specifically, the advantages of multi-attribute hashing are the following [10]:

(1) The automatic "clustering" of similar records as described above, that saves disks accesses.

(2) The address of a record is determined immediately from the record itself, without the need of consulting an inverted index or traversing a tree structure. Maintenance of an index under many insertions and deletions is a time consuming task, which is avoided with multi-attribute hashing.

(3) The method does not require merging of pointer lists on partial match queries. Moreover, the amount of work to answer such a query decreases exponentially with the number of attribute values specified in the query [32]; in contrast, for the inverted file method the work increases because of the increased number of pointer lists to merge.

(4) Used in conjunction with an ICM, multi-attribute hashing will accelerate the materialization of views, because the tuples of interest will be nicely clustered and will be retrieved in few disk accesses.

(5) Even in the case where re-execution is necessary, multi-attribute hashing will probably save disk accesses over a random placement of tuples. These savings will be more obvious if the re-execution involves joins.

4.4. Indexing of rules

Another subject of interest is how to search the intension of the knowledge efficiently. As mentioned in the previous section, resolving a task can be done either with backward or with forward chaining. The problem of quickly finding the rules that need be applied on a given situation constitutes the problem of indexing the rule base. In the case of backward chaining, the index is defined on the head of each rule. Most methods up to now build an index on the predicate name [11] or use superimposed coding techniques [31], to take the arguments into account. Multiattribute hashing on predicates and arguments seems promising again, exactly because it will group similar rules on the same disk page. Thus, all the relevant rules for a query will be retrieved with few disk accesses, avoiding the I/O bottleneck.

In the case of forward chaining, one is interested in finding quickly which rules satisfy a given condition. For example, given a collection of rules

```
IF <CONDITION> THEN <ACTION>
```

one has to index these rules based on their CONDITION part so that given a specific situation, all rules that are applicable can be efficiently recovered. The situation is more complicated than before, since conditions involve general expressions (such as selections or joins) instead of just predicate names and attributes. Our proposal here lies in transforming conditions on relations to geometric objects in some high–dimensionality geometric space [42,46] where relation attributes are thought of as the coordinates. Then, a given state of the database can be modeled as an object in this space (described by the values of the various attributes) and the problem of detecting applicable rules maps to a geometric intersection problem. For example, assuming a relation BOLTS(ID#,LENGTH,WITDH), a condition 0.6≤WIDTH≤0.8 and 1.0≤LENGTH≤1.5 can be represented as a rectangle in a two–dimensional space with coordinates LENGTH and WIDTH (see Figure 2). A scheme based on multi–attribute hashing or multi–dimensional trees (R–trees [13], R^+–trees [9,43]) can then be used to limit the search by rejecting quickly many non–applicable rules, thus improving the search performance and response time.

5. Implementation Status

We have implemented the access methods of a database system on the principles of ICM's. Our just finished prototype, called ADMS [34,35,38], exhibits tremendous speeds in accessing

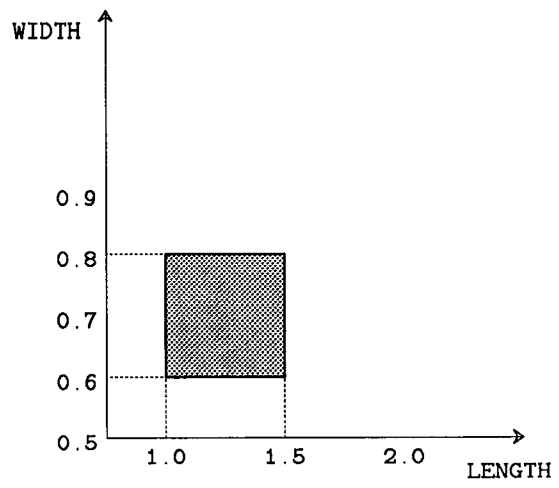

Figure 2: Rules as geometric entities

learned access paths based on incremental computing. The improvement over conventional re-execution systems ranges from four to eighty times faster, depending on the size of the increments of the utilized access paths. Our expectations were verified by the implementation:

(a) the smaller the increments, and

(b) the higher the complexity of the computation

the higher the improvement. The second characteristic is due to the fact that multilevel computations observe a lot of access path locality that subsumes a lot of the computation that ordinary re–execution models would have to repeat.

Another important characteristic of the ADMS incremental algorithms is that they are one–pass and, thus, permit interleaving of the update and cache of the access paths. This allows ADMS to produce the result much earlier than any re–execution model. Thus, the response time is very close to zero simply because ADMS starts displaying records of the previous computations while updating [38]. In simulations performed on a VAX 8600, the response to display the first record was below 2 seconds. After that, the flow of display is bounded by the speed of the terminal which is much slower than the rate of production of the incremental algorithms. This is very important for real–time systems because no existing system can come close to such response times in queries that involved two and three join operations.

Another advantage of using ADMS is that an extension of it to work in an integrated mainframe and multi–workstation environment using incremental bindings of distributed data objects has been already studied [36,37]. This is important considering the fact that expert systems implemented by means of expert database systems will be readily available for multi–user workstation based environments. The suggested prototype will be the first locally–distributed high performance expert database system.

In summary, ADMS is a very powerful database system with a sophisticated view system and access methods that are especially useful for an efficient implementation of rules, as mentioned in sections 3 and 4.

We are currently focusing on two major subjects. First, we study the theoretical foundations, the algorithms, and the design issues of the various components. Second, we have initiated an efforts towards the implementation of our ideas on top of ADMS. Among others, we study the design of knowledge base catalogs and rule and meta–rule definition languages. The multi–attribute hashing and R^+–tree access methods are under implementation. In terms of query processing algorithms we have implemented relational operators using cache techniques and have initiated an effort towards the implementation of multiple–query optimization algorithms. Finally, algorithms for the compilation and optimization of large rule bases have been devised. We expect that a working version of our prototype will be available in 18 months.

6. Conclusion

In this paper we have suggested efficient means for supporting expert systems. Our design goals can be summarized as follows

a) support large knowledge bases

b) provide very fast query processing

c) provide basic mechanisms that can be used to support various kinds of inference mechanisms ranging from simple deductive rules to general production systems.

The novelties in our approach are:

(1) The use of incremental data models. Results of previous queries are stored in cache structures which speed up tremendously the resolution of the same or similar queries in the future. ADMS, a prototype database system is now operational, and it will be used to build upon. Preliminary simulation results with ADMS show speed ups of orders of magni-

tude.

(2) Global query optimization techniques A deductive query may expand to many database queries. Traditional database systems optimize one query at a time, which does not necessarily yields a global optimal for a set of queries.

(3) Efficient methods to store and search facts and rules. We propose the use of multi–attribute hashing and multi–dimensional trees, which both cluster similar facts and rules on the same disk pages. Thus, one disk access retrieves many relevant items, avoiding the I/O bottleneck, which can cripple the performance of Expert Systems.

(4) Use of a sophisticated trigger mechanism based on update dependencies. Such a mechanism will be useful in implementing production systems.

7. References

[1] Batory, D. and Kim, W., *"Modeling Concepts for VLSI CAD Objects"*, Proc. 1985 ACM–SIGMOD Conf., May 1985.

[2] Brodie, M. and Jarke, M., *"On Integrating Logic Programming and Databases"*, in [17].

[3] Buneman, O.P., and Clemons, E.K., *"Efficiently Monitoring Relational Databases"*, ACM TODS, (**4**) 3, September 1979.

[4] Clifford, J., Jarke, M. and Vassileiou, Y., *"A Short Introduction to Expert Systems"*, IEEE Database Engineering Bulletin, (**6**) 4, December 1983.

[5] Copeland, G. and Maier, D., *"Making Smalltalk a Database System"*, Proc. 1984 ACM–SIGMOD Conf., June 1984.

[6] Dayal, U. and Smith, J.M., *"PROBE: A Knowledge-Oriented Database Management System"*, Proc. Islamorada Workshop on Large Scale Knowledge Base and Reasoning Systems, February 1985.

[7] Derrett, N.P. et al., *"An Object-Oriented Approach to Data Management"*, Proc. 1986 IEEE Spring Compcon Conf., March 1986.

[8] Duda, R. et al., *"Development of the Prospector Consultation System for Mineral Exploration"*, SRI Intern., October 1978.

[9] Faloutsos, C., Sellis, T. and Roussopoulos, N., *"Object Oriented Access Methods for Spatial Objects: Algorithms and Analysis"*, in preparation.

[10] Faloutsos, C., *"Gray Codes for Partial Match and Range Queries"*, IEEE Transactions on Software Engineering, 1987, (to appear).

[11] Futo, I. et al., *"The Application of Prolog to the Development of QA and DBM Systems"*, in [12].

[12] Gallaire, H. and Minker, J., **Logic and Data Bases**, Plenum Press, New York, 1978.

[13] Guttman, A., *"R-Trees: A Dynamic Index Structure for Spatial Searching"*, Proc. 1984 ACM-SIGMOD Conf., June 1984.

[14] Hayes-Roth, F., Invited Talk, IEEE Compcon, San Francisco, CA, February 1987.

[15] Ioannidis, Y., *"Processing Recursion in Deductive Database Systems"*, PhD Thesis, University of California, Berkeley, July 1986.

[16] Katz, R.H., *"A Database Approach for Managing VLSI Design Data"*, Proc. 19th Design Automation Conf., June 1982.

[17] Kershberg, L., Editor, *Proc. First Intern. Workshop on Expert Database Systems,* Kiawah Isl., SC, October 1984.

[18] Kershberg, L., Editor, *Proc. First Intern. Conf. on Expert Database Systems,* April, 1986.

[19] Kimura, F. et al, *"Construction and Uses of an Engineering Database in Design and Manufacturing Environments"*, File Structures and Databases for CAD: Proc. IFIP WG 5.2 Working Conf., Seeheim, 1982.

[20] Kochan, D., *"Integrated Information Processing for Manufacturing--From CAD/CAM to CIM"*, Computers in Industry (5) 4, December 1984.

[21] Kowalski, R., *"Predicate Logic as a Programming Language"*, Information Processing, North Holland, 1974.

[22] Lindsay, R.K., et al., *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project,* Mc-Graw Hill, Inc, New York, 1980.

[23] Lorie, R. and Plouffe, W., *"Relational Databases for Engineering Data"*, IBM Research, Technical Report RJ-3847, San Jose, CA, April 1983.

[24] Lorie, R. et al., *"Supporting Complex Objects in a Relational System for Engineering Databases"*, in *Query Processing in Database Systems*, Eds. W. Kim, D.S. Reiner and D.S. Batory, Springer–Verlag, 1985.

[25] Mark, L., *"Self-Describing Database Systems - Formalization and Realization"*, Technical Report TR–1484, Computer Science Department, University of Maryland, April 1985.

[26] Mark, L., Roussopoulos, N., and Chu, B., *"Update Dependencies"*, IFIP TC2 WG 2.6 Working Conf. on Database Semantics, Hasselt, Belgium, January 1985.

[27] Mark, L. and Roussopoulos, N., *"Meta-Data Management"*, IEEE Computer Magazine, (**19**) 12, December 1986.

[28] Miller, R.A., et al., *"INTERNIST-1, An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine"*, The New England Journal of Medicine, (**307**), 8, August 1982.

[29] Mylopoulos, J. et al., *"A Language Facility for Designing Database Intensive Applications"*, ACM TODS, (**5**) 2, June 1980.

[30] McDermott, D., *"R1: A Rule Based Configurer of Computer Systems"*, Artificial Intelligence, (**19**) 1, September 1982.

[31] Ramamohanarao, K., and Shepherd, J., *"A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases"*, Proc. 3rd Intern. Conf. on Logic Programming, London, 1986.

[32] Rivest, R.L., *"Partial Match Retrieval Algorithms"*, SIAM Journal of Computing, (**5**) 1, March 1976.

[33] Rothnie, J.B. and Lozano, T., *"Attribute Based File Organization in a Paged Memory Environment"*, CACM, (**17**) 2, February 1974.

[34] Roussopoulos, N., Bader, C., and O'Connor, J., *"ADMS: An Advanced Database Management System: Design Document"*, Department of Computer Science, University of Maryland, College Park, January 1984.

[35] Roussopoulos, N., and O'Connor, J., *"ADMS: Advanced Database Management System Query Language and Programmatic Interface"*, Technical Report TR–1579, Department of

Computer Science, University of Maryland, College Park, April 1985.

[36] Roussopoulos, N., and H. Kang, *"Preliminary Design of ADMS±: A Workstation-Mainframe Integrated Architecture for Database Management Systems"*, Proc. 11th Conf. on Very Large Data Bases, Kyoto, August 1986.

[37] Roussopoulos, N., and H. Kang, *"Principles and Techniques in the Design of ADMS±"*, IEEE Computer Magazine, (**19**) 12, December 1986.

[38] Roussopoulos, N., *"The Incremental Access Method of View Cache: Concept and Cost Analysis"*, submitted for publication to the ACM TODS, March 1987.

[39] Roussopoulos, N., "*Incremental Computation Models*", Department of Computer Science, University of Maryland, College Park, March 1987.

[40] Sellis, T. and Shapiro, L., *"Optimization of Extended Database Languages"*, Proc. 1985 ACM–SIGMOD Conf., May 1985.

[41] Sellis, T., *"Global Query Optimization"*, Proc. 1986 ACM–SIGMOD Conf., May 1986.

[42] Sellis, T., *"Optimization of Extended Relational Database Systems"*, PhD Thesis, University of California, Berkeley, July 1986.

[43] Sellis, T., Roussopoulos, N. and Faloutsos, C., *"The R^+-tree: A Dynamic Index for Multi-Dimensional Objects"*, Proc. 13th Conf. on Very Large Data Bases, Brighton, September 1987.

[44] Shipman, D., *"The Functional Model and the Data Language Daplex"*, ACM TODS, (**6**) 1, March 1981.

[45] Stonebraker, M., *"Implementation of Integrity Constraints and Views by Query Modification"*, Proc. 1975 ACM–SIGMOD Conf., June 1975.

[46] Stonebraker, M., Sellis, T. and Hanson, E., *"Rule Indexing Implementations in Database Systems"*, in [18].

[47] Stonebraker, M. and Rowe, L., *"The Design of POSTGRES"*, Proc. 1986 ACM–SIGMOD Conf., May 1986.

[48] Thom, J.A., Ramamohanarao, K. and, Naish, L., *"A Superjoin Algorithm for Deductive Databases"*, Proc. 12th Conf. on Very Large Data Bases, Kyoto, August 1986.

[49] Ullman, J., *"Implementation of Logical Query Languages for Data Bases"*, Proc. 1985 ACM–SIGMOD Conf., May 1985.

[50] Zaniolo, C., *"The Database Language GEM"*, Proc. 1983 ACM–SIGMOD Conf., May 1983.

[51] Zaniolo, C., *"The Representation and Deductive Retrieval of Complex Objects"*, Proc. 11th Conf. on Very Large Data Bases, Stockholm, August 1985.