Index Support for Rule Activation

David A. Brant and Daniel P. Miranker
Applied Research Laboratories and Computer Sciences Department
The University of Texas at Austin
P.O. Box 8029
Austin, TX 78713-8029
(512) 835-3381
brant@cs.utexas.edu

ABSTRACT

Integrated rule and database systems are quickly moving from the research laboratory into commercial systems. However, the current generation of prototypes are designed to work with small rule sets involving limited inferencing. The problem of supporting large complex rule programs within database management systems still presents significant challenges. The basis for many of these challenges is providing support for rule activation. Rule activation is defined as the process of determining which rules are satisfied and what data satisfies them. In this paper we present performance results for the DATEX database rule system and its novel indexing technique for supporting rule activation. Our approach assumes that both the rule program and the database must be optimized synergistically. However, as an experimental result we have determined that DATEX requires very few changes to a standard DBMS environment, and we argue that these changes are reasonable for the problems being solved. Based on the performance of DATEX we believe we have demonstrated a satisfactory solution to the rule activation problem for complex rule programs operating within a database system.

1 INTRODUCTION

The integration of rule and database systems, sometimes referred to as an *active database*, is an important step forward for both expert systems and database management systems (DBMSs). Several experimental systems have been developed (Hanson [1991], McCarthy and Dayal [1989], Sellis, Lin, and Raschid [1989], Stonebraker, Rowe, and Hirohama [1990], Widom, Cochrane, and Lindsay [1991]). It has been argued that to achieve rule and database consistency, the only viable approach is to incorporate the rules and the rule activation mechanism into the DBMS (Stonebraker [1992]). For alerters, triggers, and simple rule systems this is already being accomplished. In fact, several commercial DBMSs already provide a rule capability (INGRES [1990], Sybase [1990]). However, the computational and storage demands of alerters and triggers are trivial, compared to those imposed by medium to large scale expert systems operating within a DBMS. Perhaps this is also why so few of the database rule language projects deal with the hard problem of supporting a general *rule program* environment, one where a large number of complex rules fire in a cyclic fashion for hundreds or thousands of cycles (or perhaps indefinitely). Stonebraker has referred to these as "*hard expert systems*" and suggests that, due to the enormous demands of these systems, they will likely remain outside the DBMS for the foreseeable future. Our work is directed precisely toward supporting these complex rule programs inside a tightly integrated database rule environment. The system is called DATEX.

A critical salient feature of hard rule systems, overlooked by many, is that they are not dynamic. They usually involve an investment of several man-years for initial development. The cost and duration of problems solved by hard expert systems are commensurate with this high development cost, which implies a recurring problem with indefinite life cycle costs (Kerschberg [1987], Kerschberg [1988]). We agree, even assert, that DBMS support of hard rule systems will require some explicit built-in mechanisms and will have a permanent effect on the DBMS. The cost and duration of the problems solved leads users of these systems to provide significant resources to achieve a solution. Techniques which require additional or non-standard DBMS support are warranted and should be investigated. Moreover, these systems present an unprecedented opportunity for database optimization techniques, especially multiple query optimization. We believe DATEX demonstrates that the resources and functionality necessary to support hard problems are quite reasonable and demonstrate the feasibility of incorporating these programs into a DBMS.

Since performance is the limiting factor for these hard systems, we have not been overly concerned with the semantics or syntax of the rule language. Instead, we have developed an evolutionary sequence of rule compilers, activation methods, and execution environments. Lessons from each system were derived through the analysis of a suite of scalable benchmark programs, representative of expert systems. While the semantics of the rule language can certainly affect performance, many of the issues are orthogonal, especially if one assumes a fairly powerful and expressive language.

The key aspects of DATEX rule activation are *specialized indices*, derived from a compile-time analysis of the rule program, and *lazy matching*. The rule activation problem has been defined as determining which rules to activate and when to activate them. This problem must be addressed whenever a record is added or modified. Rule activation is assumed to include both the identification of which rules to attempt to satisfy, and finding the database records that actually satisfy those rules, i.e., the rule's *instantiations*. For hard expert systems we assume that rules may allow an arbitrarily complex join to be specified by the condition elements. In this environment, rule activation becomes quite challenging, and can easily imply many multiway joins per database update. Stonebraker [1992] suggests a taxonomy of approaches to the rule activation problem — brute force, discrimination networks, and marking schemes. Brute force is easily excluded as a viable approach for large rule programs; it simply is not computationally tractable. Discrimination networks are used extensively in expert system shells. However, we have shown that traditional Rete (Forgy [1982]) and TREAT (Miranker [1990]) discrimination net-

This work was supported in part by the Office of Naval Research, Grant N00014-90-J-1366, the ARL:UT Internal Research and Development Program, and the State of Texas.

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0042...$1.50

works are not appropriate for large scale expert systems due to their excessive space and time requirements, and that they are consistently outperformed in both time and space by LEAPS, our lazy matching technique (Miranker and Brant [1990], Miranker, Brant, Lofaso, and Gadbois [1990], Brant, Grose, Lofaso, and Miranker [1991]). The fundamental weakness of state-saving discrimination networks like Rete and TREAT is that they do not scale well to solve large, data-intensive problems. The space required by their saved state information, and the time to process it, tends to grow as a polynomial of the size of the base relations (Brant, Grose, Lofaso, and Miranker [1991]). LEAPS avoids much of the work and storage done in Rete and TREAT by using reentrant, update sensitive, data streams in rule activation as opposed to eagerly expanding a rule's entire search space during rule activation. In doing so, LEAPS is inherently a marking scheme.

Marking has been used effectively to provide high performance in other database rule systems (Stonebraker., Rowe, and Hirohama [1990]) and is intrinsic to the lazy matching technique and DATEX. Marking provides an extremely efficient way to identify the satisfied rules corresponding to a database update and is usually implemented by bit encoding at the base relation record level. In DATEX however, *indirect markers* of several varieties are used. First, an index is built on the join attributes of the rule's conditions, and markers are kept with the index records to identify which database records satisfy the constants in the conditions. This is most appropriate for rule systems that change infrequently. We call this a *filtering index*. Second, an ordered set of tuple identifiers is kept to indicate the database records that need to be processed for rule activation. The ordering is based on a given search heuristic. And, third, an ordered set of search vectors, composed of cursors, is kept for each non-empty data stream. These keep the current entry point for each partially expanded search space corresponding to a rule activation data stream.

In evaluating the efficiency of DATEX we believe that the usual database cost metrics, storage space and disk accesses, continue to be the correct measures. These metrics have been used to guide our design and experiments. We first present an architectural overview of DATEX in section 2, followed by a description of, and a rationale for, the index used in rule activation. Section 4 presents the performance results and compares them to main-memory implementations. Section 5 summarizes our contributions and describes current and future directions.

2 DATEX ARCHITECTURE

Figure 1 shows the DATEX architecture. While the current rule language of DATEX is OPS5 (Forgy [1981]), an SQL derived

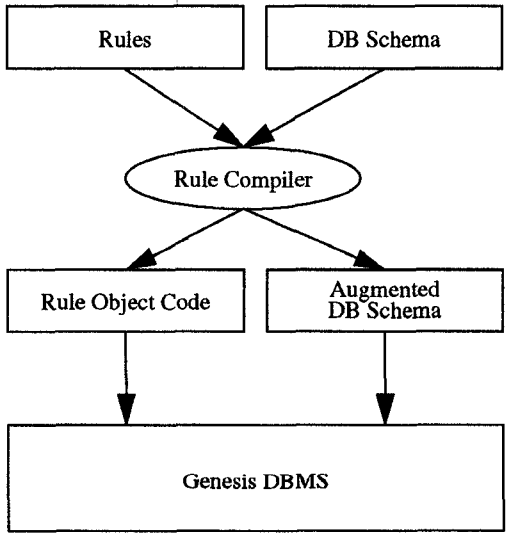

FIGURE 1. DATEX Architecture

language with object-oriented extensions has been specified and is being implemented in the next version. In order to facilitate this change, the rule language compiler is designed to create and operate on an intermediate form that is language independent. Thus, multiple rule language front-ends can be easily accommodated. The current underlying DBMS is Genesis, an extensible database system that provides a high degree of flexibility in a rapid prototyping environment (Batory et al. [1988]). It was originally thought that the rule activation techniques used in DATEX would require extensive hooks and modifications to the DBMS. However, it turns out that few extensions are needed to the standard relational DBMS in order to provide support for DATEX. In fact, a standard SQL interface is sufficient for correct behavior and the version now in development is based on Oracle. We note that the use of parallel query processing and lightweight-process communication will most likely be needed in the objective DATEX system. This is being explored in conjunction with the work reported on in this paper.

The rule compiler takes the rules and the schema as input and generates the object code for the rules and an augmented schema as output. The schema is augmented to support the efficient processing of the rule activation mechanism and is specific to the rules that were compiled. That is, the changes to the schema are rule dependent. The augmented schema is then used by Genesis when processing the rules.

The schema is augmented in several ways. They are:

- Create a *filtering index* on the *augmented* base relations. This index is presented in detail in the next section.
- For each base relation referenced by a rule condition, augment it by adding a timetag attribute. These relations are then stored in order by the value of the timetag. Timetags are strictly increasing integers assigned when records are added or modified. This is unnecessary if the DBMS assigns unique tuple identifiers.
- For each augmented relation that is referenced in a negated condition, create a second relation which will *shadow* the original. Shadow relations are only needed to support negation and their use is described in Miranker, Brant, Lofaso, and Gadbois [1990]. In a version-based DBMS, even this extension is not needed.

The compiled rules and the augmented schema are then used by Genesis to activate the rules. The primary database support for rule activation is the filtering index.

3 A FILTERING INDEX TO SUPPORT RULE ACTIVATION

Rule activation in DATEX is supported through an index designed to provide a value-based join path combined with specific information pertaining to the constant tests within conditions of a rule. The design of the filtering index has been guided by several types of information which may be used to accomplish efficient rule activation. Much of the work that has been done in the expert system domain is applicable (McDermott, Newell, and Moore [1978], Gupta [1987], Miranker [1990], Brant, Grose, Lofaso, and Miranker [1991]). At a minimum, we may learn from the in-depth studies that characterize the rules in hard expert systems. From these studies we know that for a typical rule program:

1. from one rule firing to the next, most of the database is unchanged,
2. most rule actions result in changes to the database (but the number of changes is small),
3. rule actions that modify existing records are much more common than those which add new records,
4. the changes have a small scope (few rules affected),
5. at any given time, only a small subset of rules are activated,
6. most condition elements contain constants,
7. we can effectively use the constant tests within conditions to identify rules that are NOT active,
8. constant tests provide good selectivity, i.e., they are highly selective,
9. rules are composed of 2-16 conjunctive elements,
10. rules have 2 to 5 join attributes, and

11. the join selectivity on records that satisfy the constant tests is medium, i.e., about 0.5.

These pieces of information have led to various breakthroughs and improvements in rule activation within expert system shells. The first and foremost of these was the development of discrimination networks such as Rete and TREAT. These networks filter the data based on constant tests within each condition and store the results in the α-*memories*. These provide links to the base relations and as such are indices. In addition to forming an index on the base relations, the α-memories can be used to identify which rules are not satisfiable. That is, if a rule's constant tests are not satisfied by at least one base relation record, then do not perform the joins. This provides a powerful filter to determine which joins need to be performed. It has been shown that the use of TREAT provides an effective rule activation mechanism in a database context, and it is used in the Ariel system (Wang and Hanson [1990], Hanson [1991]).

While discrimination networks provide a powerful tool in rule activation, the lazy matching technique used in the LEAPS approach has dramatically surpassed them in performance. Even more importantly for databases, LEAPS uses much less space than discrimination networks for its internal state. Less state means less state maintenance, which implies faster state maintenance. When replacing a discrimination network with LEAPS it was recognized that constant tests would still be required for any efficient rule activation since they provide such a powerful filter on the joins to be performed. Therefore, we needed to keep the information provided by the α-memories of the network. However, it wasn't at all clear that the α-memories themselves were needed. This issue led to the development of a filtering index (FI).

Even though α-memories provide an index to the base relations, they may be of little advantage in computing a join on those records, since they are not organized by join attribute value. One solution would be to implement the α-memories as a file structure and build a join attribute, value-based, index on top of them. This was the solution used in Ariel. However, since updates are so common in these systems, we argue that the overhead to support join indices on top of α-memories is not justified. Instead, we looked for a technique that would involve a single index, which would require no more entries than α-memories, and still provide both constant test filtering and join support. The result is an attribute-based index clustered by values on a given attribute. The FI file records are defined as (rel, attr,val,tt, s_bit_0,...,s_bit_{31}), where *rel* identifies a base relation, *attr* identifies a join attribute in *rel*, *val* is a value for *attr*, *tt* is the timetag of the record in *rel* having the value *val*, and s_bit_0,...,s_bit_{31} is a bit string defined below.

In discrimination networks, when a tuple is inserted into a base relation, a sequence of constant tests are performed, one test for each constant that appears in the rules' conditions. For FI records the constants are enumerated and mapped to a bit vector (s_bit_0,...,s_bit_{31}). Whereas discrimination nets employ a linear (in the number of rules) method to update α-memories, DATEX uses a log time algorithm (in the number of constants in the rule source) to determine the bit string (Nishyama et al. [1992]). In general, a record is added to the FI file for each join attribute of that tuple. The purpose of the bit string is to avoid accessing base relation records that do not satisfy the constant tests in a given predicate during the join operation, and thus replace the function of the α-memory[1]. Through analysis of our benchmark suite it was determined that 32 different constant tests for each base relation are sufficient. The FI index records are identical to any standard index, with the exception of this additional bit vector (one word).

3.1 Join Access Path Analysis

An analytical comparison was performed to determine the expected performance of the two indexing alternatives, one defined for α-memories (the AMEM file) and one for the FI. The result was that there is no case where the AMEM performs better than the FI. Moreover, the FI performs much better than the AMEM for most cases.

To investigate the performance of the FI compared to an α-memory index, we considered a file structure based on α-memories. The AMEM file has records defined as (cond, rel, tt), where *cond* identifies a specific condition of a rule, *rel* is the base relation on which *cond* is defined, and *tt* is the timetag of a record in *rel* that satisfies *cond*.

For ease of representation we assume a single file, called the BR file, contains the base-relations. Records in the BR file have the form (rel, tt,attr,val,...,attr,val), where *rel* is a base relation, *tt* is a timetag indentifying a unique record in *rel*, and the *attr,val* are the attribute/value pairs for the record.

Figure 2 shows an example of a two-way equijoin based on an AMEM to BR file access path. The join is performed in a nested loop manner by order of timestamps. As can be seen in the figure, a total of 12 records are read to accomplish the join. All records from the AMEM file must be read and, for each of those, their corresponding base-relation file record must also be read. The only parameter restricting the number of records read is the selectivity of the constant tests.

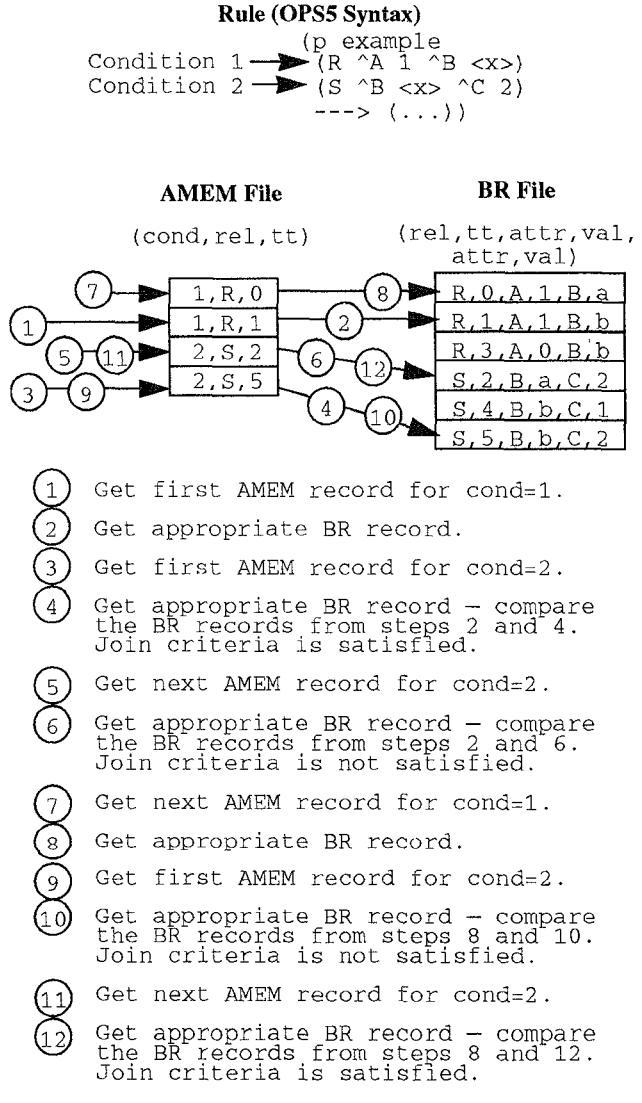

FIGURE 2. Joining relation R with relation S using AMEM file

1. We note that while the functionality is replaced, the access paths may have different lengths.

Figure 3 shows the same example as that in Fig. 1 using the FI file instead of the AMEM file as an index on the BR file. In this example, two fewer records are read from the BR file. The key functional difference is that the FI file can filter on both the selectivity of the constant tests and the join values. The actual performance of each method is subject to many factors. However, the two dominant parameters are select selectivity and join selectivity.

The overriding cost factor in using either index is the number of disk accesses incurred during a join. This is measured by examining the cost to equijoin one tuple, T, with a relation, R, using the previously defined file structures. The following parameters are needed:

- n = number of tuples in R
- σ_{sel} = constant test select operator selectivity for R in the query being executed
- σ_{join} = join operator selectivity, T join R, in the query being executed
- b = block size in bytes = 8192
- t_{FI} = FI file record size in bytes = 20

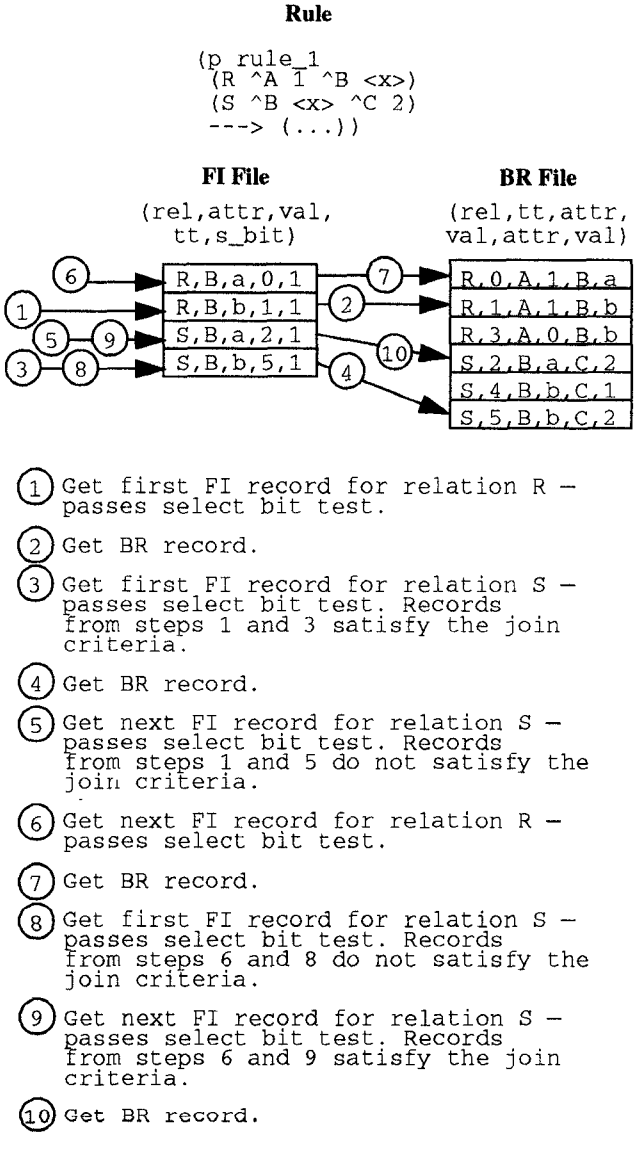

FIGURE 3. Joining relation R with relation S using FI

- t_{AMEM} = AMEM file record size in bytes = 8
- t_{DB} = BR file record size in bytes = 1024 (this is standard for many expert systems)
- f = blocking factor for B+ tree = 0.7
- B_{FI} = number of blocks for the FI file =

$$\frac{n \times t_{FI}}{b \times f} = \frac{n \times 20}{8192 \times 0.7} = 0.0035n$$

- B_{AMEM} = number of blocks for the AMEM file =

$$\frac{n \times t_{AMEM}}{b \times f} = \frac{n \times 8}{8192 \times 0.7} = 0.0014n$$

- B_{DB} = number of blocks for the BR file =

$$\frac{n \times t_{DB}}{b \times f} = \frac{n \times 1024}{8192 \times 0.7} = 0.1786n$$

Using the AMEM/BR access path, we scan the records of the AMEM file that have a condition identifier corresponding to the appropriate condition element for the query being processed. Assuming a cold start, the number of disk accesses for scanning the AMEM file is

$$min(\sigma_{sel}n, 0.0014n).$$

Thus, for all but the smallest values of σ_{sel}, all $0.0014n$ blocks will be read. These will result in

$$min(\sigma_{sel}n, 0.1786n)$$

disk accesses to the BR file. The total number of disk accesses for the AMEM/BR path is

$$min(\sigma_{sel}n, 0.0014n) + min(\sigma_{sel}n, 0.1786n).$$

Using the FI/BR access path, we scan the records of the FI file that satisfy the equijoin value from tuple T. Again assuming a cold start, the number of disk accesses for scanning the FI file is

$$min(\sigma_{join}n, 0.0035n).$$

These will result in

$$min(\sigma_{sel}\sigma_{join}n, 0.1786n)$$

disk accesses to the BR file. The total number of disk accesses for the FI/BR path is

$$min(\sigma_{join}n, 0.0035n) + min(\sigma_{sel}\sigma_{jon}n, 0.1786n).$$

Clearly, the dominant factor in these equations is the number of accesses to the BR file. Therefore, the FI/BR path is favored whenever

$$min(\sigma_{sel}\sigma_{jon}n, 0.1786n) < min(\sigma_{sel}n, 0.1786n).$$

That is, the FI/BR path can never be worse than the AMEM/BR path and will be considerably better for low values of σ_{sel} and σ_{join}. Figure 4 shows a plot of the performance advantage FI/BR over AMEM/BR. The x-axis is Join Selectivity (σ_{join},) the z-axis is

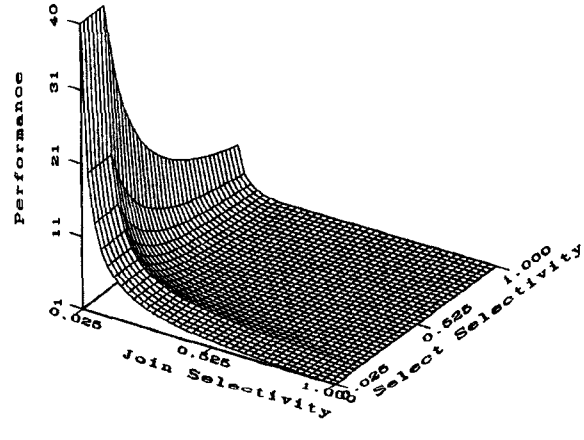

FIGURE 4. Disk Access Comparison of FI File and AMEM File

Select Selectivity (σ_{sel}), with the advantage factor on the y-axis such that

$$y = \frac{min(\sigma_{sel}, 0.1786)}{min(\sigma_{sel}\sigma_{join}, 0.1786)} = \frac{min(z, 0.1786)}{min(zx, 0.1786)} \begin{pmatrix} 0 \le x \le 1 \\ 0 \le y \le 40 \\ 0 \le z \le 1 \end{pmatrix}.$$

A particular y-axis value corresponds to the performance increase factor of FI over AMEM, i.e., a value of 10 means that FI is 10 times faster than AMEM. As can be seen from the graph, the FI/BR path is favored for most values of σ_{join} and σ_{sel}. For moderate to low values of σ_{join} and σ_{sel} the performance factor is pronounced and no combination favors the AMEM/BR path. This analysis led to the adoption of the FI file as an index for DATEX.

The overall size of the index and number of updates required to process a database change was also examined. A comparison was made between the space requirements for an AMEM system versus an FI based system[2]. This comparison was based on the largest problem size that has been run for our scalable benchmarks and corresponds to databases in the megabyte range. While these are small by database standards, they still represent an important step forward for hard problems on active databases. The object of the analysis was to examine the number of FI entries that could be expected in DATEX as opposed to the number of α-memory entries in LEAPS. The result is in Fig. 5. In general the FI approach was roughly equivalent, though somewhat better than, the AMEM in size.

4 PERFORMANCE RESULTS

The purpose for gathering performance data on DATEX was to evaluate the design decisions made during the analysis phase of the development. Important issues were execution time, FI behavior, and disk buffer hit rates. The benchmark programs and data sets used in evaluating DATEX are shown in Tables 1 and 2.

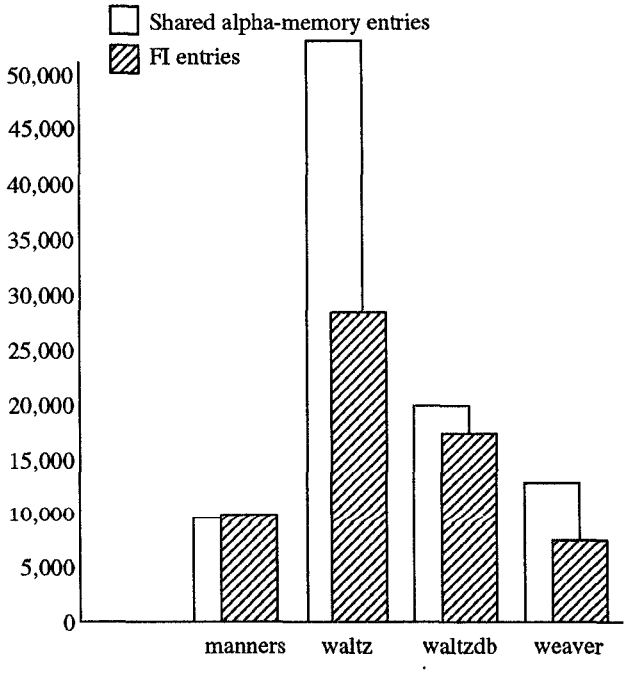

FIGURE 5. Comparison of Shared Alpha-Memory and FI Entries

2. These results assume that α-memory sharing is implemented. If not, the size of the AMEM file would be 6 to 40 times larger.

4.1 FI File Results and Analysis

One of the important aspects of using the FI file was the aggregation of the constant test select information from all relevant condition elements into a single bit mask. In the process of computing a join, this scheme could be no more effective than α-memories. However, it could be much worse. Consider the case of two different equijoins involving the same base relation. Let the condition element for join 1 have a σ_{sel} of 0.02 and let the condition element for join 2 have a σ_{sel} of 0.5. When computing join 1 using the FI file, for every tuple that satisfies the constant test, 25 tuples will be read that do not. This is the undesirable effect of aggregating the constant test select criteria. To address this concern the number of accesses made for computing all equijoins was compared with the number that passed the filter, i.e., the appropriate select bit was set. The results are shown in Table 3. For manners and waltzdb, the results were extremely good, with the number of tuples passing the filter averaging over 90%. For waltz and weaver the average was 50% to 60%. One possible explanation for the low percentage of weaver tuples passing the filter is that weaver contains a very large number of condition elements on relatively few base relations. Thus, the negative effects of aggregation are more pronounced.

Table 1. Program Summaries

Name	No. of Rules	Comment
manners	8	Finds a seating arrangement for dinner guests by depth-first search.
waltz	33	Waltz line labeling for simple scenes by constraint propagation.
waltzdb	35	A more database oriented version of Waltz line labeling for complex scenes by constraint propagation.
weaver	637	A VLSI router using a blackboard technique.

Table 2. Initial Database Size

Program	Set 1	Set 2	Set 3	Set 4
manners	16	32	64	128
waltz	864	1800	2664	3600
waltzdb	288	576	864	1152
weaver	531	791	1311	1831

Table 3. Filtering of Equijoin Buffer Accesses

Type	Set 1	Set 2	Set 3	Set 4
manners:				
Raw	1,279	7,839	53,220	384,415
Filtered	1,039	6,847	49,188	368,159
% Passed	81	87	92	96
waltz:				
Raw	62,505	126,075	184,755	248,325
Filtered	32,246	64,941	95,121	127,816
% Passed	52	52	51	51
waltzdb:				
Raw	1,779,562	5,634,112	11,642,854	19,805,788
Filtered	1,751,334	5,582,740	11,568,338	19,708,128
% Passed	98	99	99	100
weaver:				
Raw	5,864,876	8,539,384	14,934,424	22,849,464
Filtered	3,415,058	5,171,189	9,467,849	14,905,309
% Passed	58	61	63	65

The FI file is used to index equijoins only, i.e., = and ≠. All non-equijoins are produced by scanning the database directly. Therefore, no constant test selections are made *a priori* for joins on condition elements involving non-equijoins. This turned out to

have been a poor decision. Table 4 shows the number of accesses made for the non-equijoins (raw) versus the number that passed the constant test filters (filtered). By dividing the filtered amount by the raw, a σ_{sel} value can be derived. For all runs of all programs except weaver, the values of σ_{sel} were very low, ranging from 0.01 to 0.11 with a mean of 0.05. Weaver showed a σ_{sel} of 0.56. For manners, waltzdb, and weaver, the impact of having a join index for non-equijoins should be substantial, however, equijoin accesses are dominant in those programs. On the other hand, waltz is dominated by non-equijoin accesses and should demonstrate a significant speedup over the current DATEX system.

Table 4. Filtering of Non-Equijoin Buffer Accesses

Type	Set 1	Set 2	Set 3	Set 4
manners:				
Raw	818	2,922	10,970	42,426
Filtered	93	189	381	765
σ_{sel}	0.11	0.06	0.03	0.02
waltz:				
Raw	107,411	216,299	316,811	429,570
Filtered	8,559	17,035	24,859	29,464
σ_{sel}	0.08	0.08	0.08	0.07
waltzdb:				
Raw	277,499	843,343	1,714,339	2,890,487
Filtered	5,418	9,682	13,946	18,210
σ_{sel}	0.02	0.01	0.01	0.01
weaver:				
Raw	1,985,375	2,710,056	4,168,456	5,658,856
Filtered	1,085,415	1,501,178	2,323,498	3,161,818
σ_{sel}	0.55	0.55	0.56	0.56

4.2 Buffer Hit Rate

A key measure for any system based on disk-resident data is the buffer hit rate. We measured the effectiveness of the buffering for both the FI and the BR file. The FI file data (Table 5) was expected to show very high hit rates. This is due to the relatively small size of the FI file and the large number of records per disk block. The FI and BR files each had fifty 8K blocks assigned to them. This assignment was fixed for all benchmarks. The results were as expected, with the exception of waltz. As can be seen in the table, waltz had a hit rate that varied from 100% for the smallest problem, to 65.4% for the largest. While we expected that due to the fixed buffer assignment, we would see a drop in hit rates as the data set size increased, we were somewhat surprised at the magnitude of the drop in waltz. This suggests that more thought should be given to buffer management issues. The benchmarks were performed using a least recently used replacement policy. Another reactive replacement approach might make more sense, but we suspect that prefetching is likely to be effective as well. This is due to the scanning of files for executing the joins.

The buffer hit rates for the BR file are shown in Table 6. Overall these were quite acceptable. The downward trend is still present, however, and should continue as the data set increases. Simply increasing the buffer size should be both reasonable and sufficient. The typical BR file size during the execution of the largest benchmarks was approximately 1000 - 1200 blocks. Ten percent of that would suggest a buffer of 100 - 120 blocks. Therefore, for the largest data sets, our 50 block buffers were undersized by most standards.

4.3 Execution Time Results, Parallelism and Future Directions

DATEX was compared with the fastest known, main-memory, discrimination network based, expert system — OPS5.c (Miranker and Lofaso [1991]). When compared with this system (Table 7), the times range from 5 times *faster* to 74.5 times slower with an average of 22.7 times slower. The combination of LEAPS and the FI techniques nearly overcomes the overhead incurred by moving from a main-memory expert system shell to a disk-resident database.

Table 5. Disk Buffer Hit Rate for FI File Accesses

Type	Set 1	Set 2	Set 3	Set 4
manners:				
Buffer	7,092	35,303	186,300	1,099,681
Disk	2	6	25	23,990
Hit Rate	100.0	100.0	100.0	97.8
waltz:				
Buffer	791,755	2,167.149	3,928,367	6,461,663
Disk	32	440,030	1,194,672	2,238,361
Hit Rate	100.0	79.7	69.6	65.4
waltzdb:				
Buffer	4,427,726	13,177,002	26,526,927	44,504,770
Disk	5,698	225,957	591,040	1,115,422
Hit Rat	99.9	98.3	97.8	97.5
weaver:				
Buffer	9,743,095	14,075,878	24,316,040	37,104,651
Disk	19	23	37	69,808
Hit Rate	100.0	100.0	100.0	99.8

Table 6. Disk Buffer Hit Rate for BR File Accesses

Type	Set 1	Set 2	Set 3	Set 4
manners:				
Buffer	23,905	99,288	541,619	3,934,357
Disk	200	2,488	27,693	355,179
Hit Rate	99.2	97.5	94.9	91.0
waltz:				
Buffer	833,769	1,820,951	2,675,493	3,633,409
Disk	15,666	32,184	47,311	63,324
Hit Rate	98.1	98.2	98.2	98.3
waltzdb:				
Buffer	13,007.084	46,235,110	95,248,329	161,633,113
Disk	730,050	2,544,482	7,000,782	9,308,836
Hit Rate	94.4	94.5	92.6	94.2
weaver:				
Buffer	24,559,493	39,950.082	84,465,546	131,730,488
Disk	41,451	412,566	2,881,329	4,934,730
Hit Rate	99.8	99.0	96.6	96.3

Table 7. DATEX versus OPS5.c Execution Time (in seconds)

Program	DATEX	OPS5.c	Increase Factor
manners:			
Set 1	17.4	1.0	17.4
Set 2	69.3	13.8	5.0
Set 3	365.6	425.8	**0.9**
Set 4	2,640.7	15,838.5	**0.2**
waltz:			
Set 1	1,901.2	343.3	5.5
Set 2	6,939.2	988.0	7.0
Set 3	14,259.5	2,963.0	4.8
Set 4	25,102.1	3,831.8	6.6
waltzdb:			
Set 1	6,852.8	641.5	10.7
Set 2	23,443.3	2,109.6	11.1
Set 3	51,722.3	4,341.6	11.9
Set 4	83,412.7	8,033.3	10.4
weaver:			
Set 1	12,551.4	170.3	73.7
Set 2	19,055.0	255.8	74.5
Set 3	37,326.5	552.6	67.5
Set 4	58,618.0	1,053.7	55.6

A number of optimization problems remain. DATEX guarantees that every join is supported by FI entries by greedily adding new links. We do not yet have any results on finding the minimum size subset of attributes needed to establish this guarantee. Since rule predicates can be large, there is great flexibility in selecting a query plan. It is not inconceivable that an optimization method would determine that only a subset of the base relations need to be augmented with FI entries.

Although this DATEX prototype targets a uniprocessor and otherwise ignores transaction management, the philosophy of compile time analysis has been applied in a sister project on the concurrent execution of rule languages, CREL (Kou, Miranker, and Browne [1991]). The CREL system includes a parallelizing compiler. The compiler takes as input a rule program that has been written by a programmer who assumes that the entire program will be executed as a single uninterrupted transaction. The output of the compiler is a collection of rule programs, each containing a small subset of the rules in the original, such that each new rule program can be run as an independent transaction in a database environment. The concurrent execution of all the subprograms is guaranteed to be correct. The compiler is based on standard techniques presented in the literature on serializability.

Assuming an adequate parallel query processor, we extrapolate that the synthesis of DATEX and CREL will result in the parallel execution of disk-resident, gigabyte, expert system problems in elapsed times measured in minutes.

5 CONCLUSION

The DATEX system demonstrates that solving hard expert system problems in tight integration with DBMSs is not far out of reach. The critical aspects of a solution embodied in DATEX include an effective, in both time and space, marking scheme for rule activation and a specialized indexing method derived from a compile time analysis of the source rule program. We claim that acceptable performance from such an active database system is attainable only if the database contains explicit support for the inference engine. We further claim that in light of the scope of problems solved by expert systems, that we have reduced the cost of such support to an acceptable level.

Although the solutions described herein demonstrate that an active database inference engine can approach the execution speed of a main-memory expert-system shell, it is clear that additional speed is required for large databases and that the use of large-scale parallel computers may be a necessary part of an active database system.

6 REFERENCES

Batory, D., et. al. [1988], "GENESIS: An Extensible Database Management System," *IEEE Transactions on Software Engineering*, **14**:11, pp. 1711-1730.

Brant, D., T. Grose, B. Lofaso, and D. Miranker [1991], "Effects of Database Size on Rule System Performance: Five Case Studies," *Proceedings of the 17th International Conference on Very Large Databases*, pp. 287-296.

Forgy, C. [1981], "OPS5 User's Manual", Tech. Rep. CMU-CS-81-135, Carnegie-Mellon University.

Forgy, C. [1982], "RETE: A Fast Match Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, **19**, pp. 17-37.

Gupta, A. [1987], *Parallelism in Production Systems*, Pittman/Morgan-Kaufman, Los Altos, CA.

Hanson, E. [1991], "The Design and Implementation of the Ariel Active Database Rule System," Tech. Rep. WSU-CS-91-06, Wright State University.

INGRES [1990], *INGRES Version 6.3 Reference Manual*, INGRES Products Division, Alameda, CA.

Kerschberg, L. [1987], (ed.) *Proceedings of the First International Conference on Expert Database Systems*, Benjamin/Cummings Publishing Company, Inc.,Menlo Park CA.

Kerschberg, L. [1988], (ed.) *Proceedings of the Second International Conference on Expert Database Systems*, Benjamin/Cummings Publishing Company, Inc.,Menlo Park CA.

Kou, C.M., D.P. Miranker, and J.C. Browne [1991], "On the Performance of the CREL System," *Journal of Parallel and Distributed Computing*, **13**:4, pp. 424-441.

McCarthy, D., and U. Dayal [1989], "The Architecture of an Active Database Management System," *Proceedings of the 1989 ACM-SIGMOD International Conference on Management of Data*, pp. 215-224.

McDermott J., A. Newell, and J. Moore [1978], "The Efficiency of Certain Production System Implementations," In *Pattern-directed Inference Systems*, D. Waterman and F. Hayes-Roth (eds.), Academic Press.

Miranker, D. [1990], *TREAT: A New and Efficient Match Algorithm for AI Production Systems*, Pittman/Morgan-Kaufman, Los Altos, CA.

Miranker, D.P., D. Brant, B. Lofaso, and D. Gadbois [1990], "On the Performance of Lazy Matching in Production Systems," *Proceedings of the 1990 National Conference on Artificial Intelligence*, pp. 685-692.

Miranker, D.P., and D.A. Brant [1990], "An Algorithmic Basis for Integrating Production Systems and Database Systems," *Proceeding of the Sixth International Conference on Data Engineering*, pp. 353-360.

Miranker, D., and B. Lofaso [1991], "The Organization and Performance of a TREAT-Based Production System Compiler," *IEEE Transactions on Knowledge and Data Engineering*, **3**:1, pp.3-10.

Nishyama, S., K. Goolsbey, and D.P. Miranker [1992], "Optimizing Constant Tests in a Production System Environment," Tech. Rep., Dept. of Computer Sciences, University of Texas at Austin.

Sellis, T., C-C. Lin, and L. Raschid [1989], "Data Intensive Production Systems: The DIPS Approach," *SIGMOD Record*.

Stonebraker, M., L. Rowe, and M. Hirohama [1990], "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering*, **2**:1, pp.125-142.

Stonebraker, M. [1992], "The Integration of Rule Systems and Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, **4**:5, pp. 415-423.

Sybase [1990], *Sybase V4.0 Reference Manual*, Sybase Corp., Emeryville, CA.

Wang, Y-W, and E. Hanson [1990], "A Performance Comparison of the Rete and TREAT Algorithms for Testing Database Rule Conditions," Tech. Rep. WSU-CS-90-18, Dept. of Computer Science and Engineering, Wright State University.

Widom, J., R. Cochrane, and B. Lindsay [1991], "Implementing Set-Oriented Production Rules as an Extension to Starburst," *Proceedings of the 17th International Conference on Very Large Databases*, pp. 275-285.