ON THE REQUIREMENTS OF FUTURE EXPERT SYSTEMS

Ron Sauers
Rick Walsh

Artificial Intelligence Unit, Martin Marietta Corporation
P.O. Box 179, Denver, Colorado 80201

ABSTRACT

Many artificial intelligence applications require the use of expert systems. As expert systems move into new domains, several significant changes can be expected. Among these are an increase in number of rules in the rule base and an increase in the number of data elements contained in working memory.

Also, many new applications require that expert systems move into real-time domains. Here, a system must be able to process large quantities of data which are changing rapidly. Many problem-solving situations will be time critical, and the system must take into account the availability and distribution of scarce system resources.

For these reasons, the efficiency of a given expert system design and its ability to perform complex memory management tasks will become Increasingly important. This will require modifications in the traditional production system architectures. In this paper, the design requirements of future expert systems are discussed, and HAPS, a recently implemented production system architecture designed to address these issues, is presented.

I INTRODUCTION

An expert system is a computer program or set of programs capable of performing near the level of a human expert in some limited domain. Previous work in this field has produced systems which can perform medical diagnosis of blood diseases (Shortliffe, 1976), predict the physical structure of complex organic molecules Euchanan & Feigenbaum, 1978), and configure VAX-11 computer systems (McDermott, 1982). The level of success of such systems indicates that expert systems will soon move into broader, more complex domains.

As this occurs, several new constraints on the design of expert systems can be expected:

1. The system must be capable of handling larger rule bases. This is due to the fact that the level of expertise exhibited by a given expert system is directly related to the number of rules in Its rule base. Thus, If a given system Is expected to increase both its domain and level of expertise, the size of the rule base must necessarily increase.

2. The system must be capable of handling a much larger working memory set. One reason for this is that when the domain of a given expert system becomes larger, more domain-specific knowledge is required.

3. Some systems will be required to operate in real-time situations. Many new applications, such as the automation of satellite subsystems, will require the processing of data which change in real time. Also, these domains often require problem solving in time-critical situations, which means that the expert system must be flexible enough to consider constraints imposed by the availability and distribution of scarce system resources during the problem solving process.

Such constraints imply that efficiency concerns will become increasingly important in the design of future expert systems. This demands the development of new tools for the construction of expert systems which take these efficiency issues into account.

The remainder of this paper concerns the identification of areas in which the design requirements of future expert systems will strain the current production system architectures. Techniques are described which are capable of performing efficiently under these new situations. These techniques are incorporated into the design of a new production system architecture known as HAPS (the Hierarchical, Augmentable Production System), a unique expert system building tool designed to address the needs of future expert systems explicitly.

II RELATED WORK

Most of the research to date concerning the efficiency of production systems focuses on the operation of pattern matching. This is the process by which patterns from the conditional portion of a production are compared to the data elements in working memory. When all of the conditionals of a production match, that production may be instantiated. At any given time, the set of all possible production instantiations is known as the conflict set.

Pattern matching is the most time consuming operation which the interpreter must perform and is, therefore, the bottleneck of the system. The most widely known and most efficient of the

pattern matching algorithms is the Rete Match Algorithm (Forgy, 1982). This algorithm takes advantage of the following characteristics of production systems:

1. Pattern Similarity—Since productions are testing against the same set of data items, many of the patterns will have similar characteristics. Thus, at least some of the matching for many of the patterns can be done simultaneously.

2. Temporal Redundancy—The contents of working memory change slowly over Lime. On any given cycle, a few data items may be added and a few may be modified or removed, but most remain the same. Thus, pattern matching information can be saved from cycle to cycle, with only a few modifications.

Taking advantage of these concepts requires the compilation of production patterns into a discrimination network. Data elements which enter working memory are sent through at the root of the network. At the terminal nodes, modifications are made to the conflict set. This results in an algorithm where the execution time required for a single firing is, in the best case, independent oi the number of data items in working memory. For this reason, the Rete Match Algorithm is implemented in the interpreters of most state-of-the-art production systems.

I II GOAL DIRECTEDNESS

Introducing goal directedness into a production system eases the writing of programs in that system. For example, the OPS5 system provides an alternate conflict resolution strategy which facilitates means-ends analysis (Forgy, 1981). Other systems are explicitly goal-directed (e.g., Sauers & Farrell, 1982); the system builds an explicit hierarchy of goals during execution. At any given time, such a system is focused on a single goal, and the system objective is to find a way of achieving that goal.

Conceptually, productions In a goal-directed system are expressed in the form:

IN a given goal context,
IF some set of conditions is true
THEN perform some set of actions.

The use of an explicit goal hierarchy has many advantages. From the user's standpoint, goal-directed programs are easier to construct, since a production is prohibited from firing unless it is applicable to solving the specific subtask at hand. Equally important, however, Is the increase in run time efficiency which can be obtained by incorporating goal-directedness into a system design.

First, note that restricting productions to firing in a particular goal context forms natural partitions in the rule set. A set of equivalence classes is defined, where productions In the same class are all productions which fire in the same type of goal context. During system execution, we know which class of productions is relevant to achieving the current goal, and only productions in that class need to be considered for instantiation. This means that no class of productions needs to be processed until that class becomes relevant to the problem solving process.

Similarly, the nature of the goal hierarchy can be exploited in the design *of* the working memory structure. Often during the execution of a goal-directed system, data items are inserted into working memory which are relevant only to achieving a given goal. Once a method for achieving that goal has been determined, those data items are no longer needed.

We can take advantage of this characteristic for efficiency reasons by introducing hierarchical levels of working memory. Data elements can be declared local to a particular goal; when a goal is achieved, its local data elements disappear. Such a scheme allows the working memory structure to grow hierarchically along with the goal structure. This is important because it permits all processing resulting from the creation of a given data element to occur only within a limited local environment.

While the notion of goal directedness is by no means novel, the incorporation of an explicit goal hierarchy into a production system architecture is important in terms of efficiency because it provides two key capabilities:

1. Productions can be integrated into the problem solving process in such a way that system resources are spent in processing only those productions which are applicable to the solution of the current subtask.

2. A hierarchical working memory scheme can be introduced, which will allow for more efficient management of the large, domain dependent knowledge bases which will exist in future expert systems.

An explicit goal hierarchy also allows for the construction of expert systems with a much more general control structure, which more closely models the problem-solving processes of a human expert in a dynamic environment (see Section VII).

IV PRODUCTION HIERARCHIES

Much research has been done concerning the nature of the knowledge contained Inside a production. This research has led to a general distinction between productions and meta-productions. In general, a standard production represents a piece of expert knowledge specific to a given domain. A meta-production contains meta-knowledge; that is, knowledge about the system's knowledge and how to use it. The use of meta-level knowledge allows a system to make high-level decisions, such as which of a set of given solution paths is most likely to lead to the best answer to the problem at hand.

Usually, meta-productions have some sort of precedence over regular productions. This is due to the nature of the knowledge encoded in the meta-productions; it is generally preferable to make high-level decisions concerning how the system will attempt to solve a problem before considering the minute details of the solution itself.

This idea has been important in the design of state-of-the-art, hierarchical planning systems (e.g., Stefik, 1980). A hierarchical planner will first produce an abstract representation of a plan to accomplish a given task. Then, by gradually considering more constraints, this plan becomes increasingly detailed, until the final plan in complete detail is produced.

These concepts can be generalized to produce the notion of production hierarchies. In this scheme, productions are grouped together into sets such that those productions which are similar according to some pre-specified criteria are in the same set. Rules may be grouped according to such criteria as level of knowledge represented, or level of detail of problem solution produced.

Next, we provide the capability of fetching selected rule sets into the environment at execution time. The system is provided with initial rule sets by the user. Existing productions can recognize problem-solving situations which require that additional rule sets be available, and these can be loaded into the environment and declared local to a given goal. The resulting production hierarchy can grow as new levels of subproblems are identified.

This is best illustrated through an example. Suppose an expert system has the task of trouble-shooting some malfunction in a satellite system. Initially, only general problem-solving procedures and high-level troubleshooting rules reside in the environment. One production might notice that the malfunction was due to a loss of power, and might suggest focusing on the power subsystem. At this point, the system has the ability to load in a new rule set containing productions specific to power subsystems. As the malfunction becomes isolated to a smaller subset of possible faults, we might load a production set specific to the power subsystem of a particular satellite, a production set specific to solar arrays, and even a production set particular to environmental causes of solar array failure.

One advantage of this scheme is that It provides an efficient way to manage large rule bases. Individual groups of productions can reside in separate source files (and on different physical devices). A group of productions does not need to reside In memory until they are needed. Also, the addition of a new production set is often triggered by the system's attention to a particular goal. In these instances, not only can the goal hierarchy be used as a framework for building production hierarchies, but it can also serve as a framework for dismantling them. If a production set was brought into memory as a response to the creation of a particular goal, then it can be removed from memory when that goal has been achieved.

The addition of a hierarchical production scheme into the expert system environment works well in conjunction with the goal-directed partitioning strategy discussed earlier. The resulting system is one in which, conceptually, a library of production sets relevant to different problem solving tasks is available. Several sets are selected during system execution, and the goal-directed nature of the system guides the search through these selected sets. Together, these techniques provide an efficient mechanism for managing large rule sets.

V ALTERNATE MEMORY STRUCTURES

The efficiency of a data representation is usually measured along two dimensions: space and time. We have already considered the space efficiency of working memory; a memory management scheme based on hierarchical levels of working memory has been described. We still need to address, however, the time efficiency of the operations performed on working memory.

The operations standardly performed on working memory are pattern matching against the individual data elements, and updating the contents of the data base. Efficient Implementation of these operations is provided through the use of the Rete Match Algorithm previously discussed.

The Rete algorithm was explicitly designed to exploit the following characteristics of production systems:

1. Due to the fact that productions test against the same data set, many patterns will be similar, and matching against features of patterns which are the same can be done simultaneously.

2. The contents of working memory change slowly over time. Therefore, pattern matching information can be saved between match cycles to avoid redundant calculations.

Temporal redundancy is critical to the efficiency of the Rete algorithm. Imagine a situation, however, in which a set of data items changes frequently (for example, during every recognize/act cycle). Each time any data element is updated, all of the production instantiations in the conflict set which depend on that data item must be removed or tagged invalid. Then, the new value of the data item must be matched again to form the set of valid instantiations.

This is clearly inefficient; yet, this is precisely the situation which exists in a real-time environment. Real-time systems must deal with such data as health and status information, links to other real-time information networks, and feedback from sensor systems. Such data may change hundreds of times in the interval between production cycles.

One solution to the resulting efficiency problem is to provide additional, globally accessible memory structures in addition to standard working memory. A variety of these structures have been implemented in HAPS, including system attributes, arrays, and tables. Pattern matching must now occur in two stages. Matching against standard working memory remains a data-driven process; that is, matching is done at the time the data base changes. Matching against alternate memory structures, however, must be perform dynamically, at instantiation time (that is, at the start of each cycle).

In addition to solving some of the problems concerning the processing of real-time data, this scheme simplifies the interface to other software systems. This permits the development of expert systems consisting of many components, not all of which are rule based. Finally, this scheme allows for the possibility of creating separate match procedures for each data type. Thus, in future systems, the idiosynchratic behavior of each memory structure can be identified and exploited, in the same way the Rete algorithm takes advantage of the temporally redundant nature of a standard working memory.

VI CONFLICT RESOLUTION

Conflict resolution is the process whereby one production is selected to execute from the set of all applicable Instantiations. Two types of conflict resolution strategies are common: elimination strategies, and selection strategies. An elimination strategy is one which rules out the consideration of certain alternatives. Selection strategies are then used to pick among the remainder. An example of an elimination strategy is refraction, which rules out production instantiations which have already fired in the past. A common selection strategy is specificity, which selects specific productions over more general ones.

The standard conflict resolution strategies have an important flaw: they are in general unaware of the characteristics of the system environment, and are thus unresponsive to changes in this environment. This is especially true of those characteristics which directly concern system efficiency. An intelligent system would be able to maintain a set of system performance statistics over time, and would Include In the conflict resolution process strategies which enable the system to make use of these statistics.

This scheme can be demonstrated through the use of an example. Suppose a large expert system has been installed and has been operating in the same environment for a fairly long period of time. Assume also that a statistical summary of past system performance is available. Now, in a certain context, several productions are able to fire, each representing a different approach to the solution of the problem at hand. One production may have a history of consuming large amounts of CPU time, and another may have a history of seldomly leading to a satisfactory solution. It might be desirable to eliminate these productions immediately. Of the remaining rules, some might be more likely to lead to long-term solutions than others, and it may be desirable to select these first. Thus, conflict resolution can be used to help allocate resources to tasks with the greatest benefit.

Conflict resolution strategies can also address real-time constraints by enabling productions themselves to alter the conflict resolution strategies in critical situations. For example, suppose an expert system is given a limited amount of time to solve a critical problem. If time begins to run out, we may want to consider only those productions which always produce quick solutions. Although this will probably provide only a short-term failure workaround, the time-critical nature of the situation will have disappeared, and the system will now have more time available to pursue a more permanent solution.

Finally, many systems require that some operations (In this scheme, for example, the matching against alternate memory structures) be performed at instantiation time. The conflict resolution process usually assumes that it is given a valid conflict set—a set of production instantiations all of which have all of their conditionals satisfied. Here, however, the system has at instantiation time a list of candidates for the conflict set. These candidates are not valid members of the conflict set until they are found to satisfy the set of tests performed at instantiation time.

This suggests that a type of meta-conflict resolution procedure be available. Meta-conflict resolution stategies are able to consider such statistics as cost for instantiation and can eliminate selected candidate instantiations before they are tested for validity. For example, if the system is performing under a time-critical condition, it Is reasonable to immediately eliminate candidate instantiations which would require a great deal of some expensive processing (e.g., inferencing; see Section VIII) to enter the conflict set.

VII CONTROL STRATEGIES

Most production systems use the same general control structure; this is the recognize/act cycle. In goal-directed systems, there is an additional issue which needs to be addressed. The system has a hierarchy of goals which need to be achieved, and one must be chosen to be the focus of attention on each cycle.

One of the most common search strategies applicable here is depth-first search. In this strategy, a goal is pursued until it is achieved, or, if it sprouts subgoals, until all of its subgoals have been achieved. Another applicable search strategy Is the breadth-first search, in which all of the subgoals of a particular goal are expanded one level before any deeper

expansion occurs.

These types of search strategies are adequate for some applications. They do, however, have a basic inherent flaw: both of these blind search strategies are unresponsive to changes in the system environment and are, therefore, unable to take into account the particular characteristics of the problem being solved.

This flaw is an important one when we are concerned with system efficiency. As a simple example, consider the situation in which a system has two methods available to achieve a particular goal, each of which involves the achievement of a different subgoal. The system may not, however, have enough information available to be able to select the more efficient method in this particular case. Using a blind depth-first search strategy, the system may waste large amounts of some scarce system resource (for example, CPU time) pursuing one goal when pursuing the other would have led to an immediate solution.

This problem is closely related to those we have identified during the analysis of conflict resolution. Thus, it seems appropriate here to apply techniques which are traditionally reserved for conflict resolution: the use of selection and elimination strategies. We can make use of system performance statistics to produce a more effective search of the goal hierarchy, thereby allowing system resources to be applied in the directions where they will most likely produce desirable results.

For example, in order to solve the problem discussed above, an elimination strategy can be used which rules out the pursuit of goals which have consumed more than a specified amount of a given system resource. Alternatively, a selection strategy may be used which pursues those branches of the goal tree which have produced the largest amount of new information for each unit of some selected resource. Similarly, these control strategies can be used to prevent the pursuit of goals which have failed under similar circumstances in the past, and to prevent infinite recursion.

Control strategies work well in conjunction with the explicitly represented goal hierarchy. For example, the goal hierarchy may contain an explicit OR branch; that is, in order to achieve a goal, we can either achieve one set of subgoals or a second set of subgoals. This represents a situation in which there are multiple possible solution paths. The system can begin to pursue one solution path and then decide, through the use of a control strategy which monitors depth of subgoal expansion, that it might be more advantageous to switch to an alternate solution path. The resulting control structure is one which is capable of recovering from situations which human experts would be able to avoid, yet which could not be handled using more traditional production system architectures.

Another problem which is directly related to memory management is the need for inference. This need can be demonstrated quite easily. Suppose some pattern in the conditional portion of a production does not match. There are at least two reasons why this may occur:

1. The data item being tested for represents a proposition which is false in the current context.

2. The proposition is true in the given context, but is not explicitly represented in working memory.

In the latter situation, we may be able to infer the truth of the desired data element from other data items which are explicitly represented. In this case, an interpreter which does not permit inferencing will disallow the execution of what we would like to be a valid production instantiation.

Many knowledge representation schemes provide automatic inferencing capabilities (e.g. Genesereth, Greiner, & Smith, 1980). In these schemes, inferencing is performed at the time the data base is queried. Inference mechanisms such as these usually require the representation of some form of meta-level knowledge. These schemes, however, are not currently applicable in the production system scenario. This is mainly due to the fact that we can take advantage of properties such as temporal redundancy only for data items which are explicitly represented in memory.

For this reason, inferencing in the production system scenario is an expensive operation. One method for increasing its efficiency is to incorporate the following into the inference scheme:

1. The individual patterns on which Inferencing is permitted should be tagged. This ensures that inferencing is permitted only on those clauses for which it is desirable.

2. Inferencing is performed only when the appropriate data elements required for the instantiation of a chosen production are not explicitly represented in memory. Furthermore, this inferencing is delayed until instantiation time and is only executed for instantiations which have passed the meta-conflict resolution process.

Under this scheme, we can associate a cost with any given inference mechanism, and potential instantiations which would require extensive calculations during the test for validity can be eliminated by meta-conflict resolution strategies.

Finally, it has been shown that the inference procedures required to derive new data elements are not uniform over all data types (e.g., Fox, 1979). Thus, it is desirable for the user to be permitted to define external inference routines and to indicate which types of data element these procedures are designed for.

Combining these ideas results in a simple

inference scheme which provides many of the advantages of automatic inferencing mechanisms without imposing an unnecessary strain on system resources.

IX CONCLUDING REMARKS

The research which has been described in this paper has identified some of the problems which will be faced by the designers of future expert systems. It has also suggested several design constraints to be considered in production system architectures designed to handle these new expert systems.

These considerations have led to the design of a new production system architecture known as HAPS, the Hierarchical, A.ugmentablc Production syystem architecture. HAPS is a goal-directed system, which allows both hierarchical levels of working memory and the dynamic construction of production hierarchies. In addition, HAPS provides predefined global memory types designed to facilitate the implementation of large expert systems in real-time situations.

The system also provides modular, modifiable sets of control strategies and conflict resolution strategies which make the system responsive to changes in its environment. These strategies take into account cost estimates, history of system statistics, and availability of scarce system resources in order to guide the problem solving process more effectively.

User-declared Inference procedures are also handled, and cost estimates of operations performed at Instantiation time can be included in meta-conflict resolution strategies.

Finally, the system is equipped with a sophisticated production compiler designed to increase the overall level of system efficiency. It Is hoped that systems such *as* HAPS will greatly facilitate the building of the types of expert systems which can be expected in the near future.

REFERENCES

[1] Buchanan, B. G. and E. A. Fiegenbaum, "DENDRAL and META-DENDRAL: Their Applications Dimensions", Artificial Intelligence 11, 5-24, 1978.

[2] Forgy, C. L., On the Efficient Implementation of Production Systems, Ph.D. Thesis, Carnegie-Mellon University, 1979.

[3] Forgy, C. L., 0PS5 User's Manual, Dept. of Computer Science, Carnegie-Mellon University, 1981.

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

[5] Fox, M. S., "On Inheritance in Knowledge Representation", In Proc. Sixth IJCAI, pp. 282-84, 1979.

[6] Genesereth, M. R., R. Greiner, and D. E. Smith, MRS Manual, Stanford Heuristic Programming Project, Memo HPP-80-24.

[7] McDermott, J., "RI: A Rule-Based Configurer of Computer Systems", Artificial Intelligence 19, 39-88, 1982.

[8] Sauers, R. and R. Farrell, GRAPES User's Manual, Technical Report ONR-82-3, Carnegie-Mellon University, 1982.

[9] Shortliffe, E. H., Computer-Based Medical Consultations: MYCIN, American Elsevier, New York, NY, 1976.

[10] Stefik, M. J., Planning with Constraints, Ph.D. Thesis, Stanford University, 1980.