Bounding the computation time of forward-chaining rule-based systems

Kevin Tomsovic\textsuperscript{a} and Chen-Ching Liu\textsuperscript{b}
\textsuperscript{a}School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752, USA
\textsuperscript{b}Department of Electrical Engineering, University of Washington, Seattle, WA 98195, USA

Received 2 March 1990

Abstract


For real-time applications of expert systems, success depends on the computational efficiency of the implementation. In this study, we propose an analytical method for evaluating the processing time of forward-chaining rule-based systems. An upper bound based on this system model is developed. If the upper bound stays within the time available for planning the operational or control task, the expert system would be able to complete the rule-processing in time. To compute the upper bound, the worst case working memory element sets are obtained for each functional step of the matching procedure. The worst case time for rule selection in the conflict resolution step is also derived. The maximal number of firings for each rule is considered in order to arrive at a bound for total processing time. Numerical examples are presented which point out the importance of rule and data structures in the efficient implementation of rule-based systems.

Keywords: Discrimination net; expert systems; forward-chaining; knowledge-based; pattern matching; power systems; rule-based; RETE algorithm; voltage control.

1. Introduction

Expert Systems (ES) have attracted interest in recent years with their potential for encoding large amounts of domain-specific knowledge. The fundamental idea is to separate knowledge of the application domain from the inference procedure and data in order to allow incremental improvements within the knowledge base. This effectively simplifies development. Typically, knowledge is encoded in rules, i.e. productions, or other highly structured representations (e.g. frames). The potential areas of application are extensive [10]. Among the best known of the early ES is R1 [14], developed for configuring VAX computers. Further work on R1 has suggested the importance of a highly structured knowledge base to minimize maintenance effort and to ensure accurate results [3]. The application of ES in on-line environments is a new area of research. By 'on-line', it is meant that the ES is performing in an operational environment, for example, operating a computer system [8], monitoring and restoring a power utility network [22], and identifying hazardous weather conditions for air traffic [4], where the ES needs to perform tasks under time
constraints. This study deals with the analysis of computational aspects of rule-based systems as on-line operational aids.

Current work on ES has focused primarily on specific applications and new representations, including the use of frames and object-oriented programming. Up to the present, little has been reported on the analytical aspects of the computational efficiency of ES. Implementation greatly determines whether a given rule-based program is efficient enough to solve a specific class of problems. Notable implementation issues include rule structure, data structure, software language and hardware support. One of the initial efforts in efficient processing of rule-based systems was the RETE algorithm which utilizes a fast pattern matching method [6]. Improvements on the RETE algorithm have been proposed [17]. Other work has focused on parallel pattern matching and concurrent rule firing [1, 5, 7, 15, 16, 18, 20]. A number of ES shells contain efficient algorithms for implementing production systems [10]. These studies all concentrate on improving the efficiency of ES development tools. A hardware chip performing approximate reasoning has been developed as a means of improving efficiency [21].

Literature on analysis and design guidelines for efficiency of rule-based systems is scarce. Reference [2] provides some general guidelines for efficient implementations of OPS5 applications, i.e. to avoid bottlenecks in the rule processing. The lack of literature is due partly to the difficulty of formalizing the productions for analysis. Furthermore, the rule-chaining process is non-procedural and, therefore, finding the bottleneck or, the most time-consuming rule-chain, involves a sophisticated search process.

Thus, this study is concerned with systematically evaluating computational performance of a developed ES and generating criteria for performance improvement. Specifically, we analyze the computation time of forward-chaining inference engines based on a very efficient algorithm, RETE [6]. OPS5 and OPS83 are among the production system languages that make use of the algorithm. The proposed analytical approach is, in general, adaptable to other rule-based languages or chaining techniques.

For on-line applications, it is important to know the response time of an operational aid to a contingency (event). If the ES can respond effectively in the worst-case then it is guaranteed that the ES can respond within required time limits. Determining the actual worst case or the most time-consuming case is normally infeasible. Thus, an upper bound on the computation time required for any scenario of the problem domain is derived.

Practically, this study was motivated by the author's observations of the wide variance in computation time that could arise from minor changes in rule base design. Clearly, different rule base implementations can represent the same functional knowledge while implementing different search processes (e.g. employing a breadth-first search over a depth-first search). Variations in efficiency, however, may also arise in less direct ways. For example, a diagnostic expert system can contain rules to determine a system failure based on the status of measurements from the sensors. The conditions leading to a system failure can be written in different ways; specifically these conditions can be reordered without changing the knowledge content. Since the inference procedure is designed to match the data to rules, the matching can vary depending on the rule structure. Here, a poor choice in the ordering of rule conditions could lead to unnecessary computation. An upper bound on processing time can help expose such discrepancies.

This paper first presents a system model for the forward-chaining mechanism that is used in the RETE algorithm. The input to the system model is the set of working memory elements and the output is the action or conclusion suggested by the ES. Each step of the pattern matching along the discrimination net structure is modeled according to its intended function. To obtain the upper bound, the worst case working memory element sets are obtained for each functional step of the matching procedure. The worst case time for the
Conflict resolution is also derived. The maximal number of firings for each rule is considered in order to arrive at an overall bound for the processing time.

The conditions required for the derived upper bound to be hard (achievable) are identified. Issues involved in decreasing computational time are discussed. Improvements are implemented and numerical examples are used to show the practicality of this bound. Some preliminary results are presented in [24].

2. Functional model for forward-chaining mechanism

The execution of an ES differs from that in a procedural language such as FORTRAN. Figure 1 illustrates how a forward-chaining rule-based program is executed. There are three major steps: rule-matching, conflict resolution, and act (rule-firing) [6].

The system state is stored in working memory (WM) according to a specific data structure. For the situation represented by the patterns in WM, the complete set of rules that are satisfied by WM must be identified. This step is called rule-matching. Conflict resolution selects the most applicable rule in the matched rule set (conflict set) according to the conflict resolution scheme. The action part of this rule will be executed. This action may initiate changes in the data set, and therefore, the match-conflict resolution-act cycle would be repeated. So, one iteration consists of a matching process based on a given set of data changes, an evaluation procedure to select the most applicable rule and action step which directs any I/O or external procedures and determines the set of data changes for the repetition of this cycle. The iteration stops when no more rules are satisfied or an explicit halt is encountered.

ES may be quite slow since the rule base must be searched before each rule firing. It is important to identify those aspects which slow down the program execution [2, 9]. Due to the complexity of large rule-based systems it is difficult or impossible to obtain an exact expression for the processing that must be performed during execution. In this section, we propose a system-oriented model which provides an analytical foundation for determination of an upper bound.

2.1 Domain characteristics

The ES knowledge of the domain problem is founded on a set of attributes:

\[ A = \{a_1, \ldots, a_m\} \]
where \( m \) is the number of attributes required to characterize the given domain. Subsets of the attributes, \( A_i \), are used in combinations to describe the system and may take on numerical or symbolic values. Furthermore, the attributes are constrained by a set of allowable values. Such constraints, represented by a mapping \( F \), define the possible scenarios (system states), \( \Omega \):

\[
\Omega = F\left( \prod A_i \right) \quad A_i \subset A
\]

(1)

where \( \prod A_i \) indicates products of attribute subsets. Clearly, knowledge of these domain constraints is fundamental to analysis of the ES performance. A particular operating condition, \( \Omega_0 \), is described by the function, \( f_0 \):

\[
\Omega_0 = f_0\left( \prod A_i \right) \quad A_i \subset A.
\]

(2)

Thus, \( F \) represents a set of possible mappings and \( f_0 \) represents a specific mapping. For example, let \( a_1 \) be a circuit protection device, say, a fuse or a circuit breaker (CB), and \( a_2 \) the operating limit of such devices, say, either 10 amps or 25 amps, then together, \( a_1 \) and \( a_2 \) describe the operating limits of either a fuse or a circuit breaker. Further, suppose all fuses have a capacity of 10 amps while a CB may be of either size. Then in the notation of (1) and (2):

\[
A = \{(\text{fuse, CB}), (10 \text{amps, 25 amps})\}
\]

\[
\Omega = \{(\text{fuse}), (\text{CB}), (\text{fuse, 10 amps}), (\text{CB, 10 amps}), (\text{CB, 25 amps}), (10 \text{amps}, 25 \text{amps})\}.
\]

The elements of \( A \) are possible values for attributes and a specific condition \( \Omega_0 \) must lie in \( \Omega \). If a CB of 10 amp capacity is the relevant device then:

\[
\Omega_0 = (\text{CB, 10 amps}).
\]

2.2 Rule model

The rule base defines the problem solving capability of an ES. Consider the set of rules:

\[
\{R_1, R_2, \ldots, R_n\}
\]

where \( n \) is the number of rules. (Throughout the following development, the distinction between a rule and a particular instantiation of a rule is implicit within the network representation and will be emphasized only where confusion may arise.) For \( R_i \), we will represent its condition part as:

\[
f_i\left( \prod A_i \right) \in \Omega_i \quad A_i \subset A
\]

(3)

where \( f_i \) is a function of attributes and \( \Omega_i \) is the applicable operating condition(s) for \( R_i \). Notice that in general \( \Omega_i \) can cover several operating points. If \( f_i \) is broken down into tests on individual attribute subsets, we have:

\[
f_i\left(A_{i1}\right) \in \Omega_{i1}, f_i\left(A_{i2}\right) \in \Omega_{i2}, \ldots, f_i\left(A_{il}\right) \in \Omega_{il},
\]

(4)
and
\[ \Omega_{i1} \times \Omega_{i2} \times \cdots \times \Omega_{ii} = \Omega_i, \]
\[ A_{i1} \times A_{i2} \times \cdots \times A_{ii} = \prod A_i. \]

There are \( l \)-conditions to be tested in order for \( R_i \) to be satisfied. Each rule defines actions upon execution of the rule, so that:
\[ f_i\left(\prod A_i\right) \Rightarrow d_i\left(\prod A\right). \]

Here, implication \( \Rightarrow \), is used to represent that the execution of \( R_i \) results in \( d_i \), a decision which changes the value of attributes and/or performs some external function(s). External functions will be discussed later. For now, consider the changes in attribute values. These changes will depend on the present attribute values. In our notation:
\[ A_{i1}^R = d_{i1}\left(\prod A\right), A_{i2}^R = d_{i2}\left(\prod A\right), \ldots, A_{im}^R = d_{im}\left(\prod A\right), \]

where the superscript \( R \) indicates resultant values.

### 2.3 Fast pattern matching and working memory

Typically, the most time-consuming step is matching the rule patterns, i.e. determining all rules such that (3) is satisfied. Observe two characteristics of production systems which can be used in order to gain efficiency [6]. First, many rules have identical conditions, so the condition checking may be shared in order to avoid repeating the same operation. This is accomplished by compiling rules into a network, called a discrimination net. For example, in Fig. 2, if \( R_1, R_2 \) have identical conditions, say \( f_{11}(A_{11}) = f_{21}(A_{21}), f_{12}(A_{12}) = f_{22}(A_{22}) \) and \( \Omega_{11} = \Omega_{21}, \Omega_{12} = \Omega_{22} \), the condition checking operations will not be repeated. Second, typically only a small number of changes are made to WM for each rule firing. Thus, the results of condition checks can be stored and only changes to working memory processed.
That is, if \( A_{ij}^k = A_{ij} \) no condition check on \( A_{ij} \) should be repeated. Note for large amounts of data or when conditions are varying more rapidly such an approach will be ineffective. In such cases, traditional database search techniques may be more relevant. However, the connection between databases and knowledge-bases is beyond the scope of this paper. Several references are available (e.g. [26]).

For clarity and efficiency, the domain attributes are organized into some data structure. This structure greatly determines the processing required to check the constraints in (4). To begin, attributes are grouped to form classes in WM. That is, the \( i \)-th WM class, \( \text{WMC}_i \), is represented by a set of attributes \( a_i \in A_i \). A separate condition of a rule tests attributes in one of these classes only. In the above notation, each \( A_{ij} \) corresponds either to one of these WM attributes or some set of attributes for matching among WM classes.

A working memory element (WME) \( w_i \) belongs to some WM class with values assigned to each of its attributes. The problem state is described from the set of WMEs:

\[
\text{WM} = \{ w_i \} \quad i \in 1, 2, \ldots
\]

The state of the discrimination net is represented by WME sets. In Fig. 3, a net consists of a number of nodes represented by various conditions on attributes. Two types of operations are performed before a rule is matched to the WM set: attribute testing – finding the sets of WMEs which satisfy conditions on attributes, e.g. \( a_1 = 1 \), \( a_4 = \text{open} \), and \( a_3 > 1 \) in Fig. 3, and intercondition testing – cross checking to identify WME combinations that satisfy conditions at a joint node, e.g. \( a_3 = a_6 \) and \( a_7 = a_8 \). At the nodes represented by \( a_1 = 1 \) and \( a_4 = \text{open} \) (on the same path), WMEs in the same class need to be checked. Therefore, the results of attribute testing will be a subset of WM, i.e.

\[
\{ w_i \in \text{WM}: w_i \in \text{WMC}_1, a_1 = 1, a_4 = \text{open} \}.
\]

At a joint node, the intercondition testing results in a set of WME combinations (the WMEs may or may not be from the same class) which satisfy their respective constraints.

---

**Fig. 3.** Pattern matching with WMEs.
Bounding the computation time

Hence, the combined elements have the form of:

\[ w_i \times w_j \in WM \times WM. \]

This would be the case for the node represented by \( a_5 = a_6 \) in Fig. 3. Note, that the cross product increases its dimension with depth in the network. This increasing dimension is indicative of increasing computational costs for pattern matching. However, negative nodes (i.e. joint nodes where only if no WME matches a condition can the rule be satisfied) are a special case. Since a negative node is satisfied when no WME combinations are matched, it does not increase the number of WME combinations after passing such a node. Based on the above discussion, the state of the discrimination net is represented by WME sets, i.e.

\[ N_j = \{ n_i \}, \quad n_i \in X_d WM, \]

where \( X \) represents the product space over WM. For example in Fig. 4, if \( d = 2 \) then \( n_i \in WM \times WM \). \( N_j \) represents node \( j \) in the discrimination net and \( n_i \) defines the WME products which have been matched at that node. A WME \( w_i \) encounters \( N_j \) if all tests along a path from the root node to \( N_j \) are satisfied. At a terminal node (the final condition check in satisfying a rule), each \( n_i \in N_j \) leads to an instantiation of the rule \( R_j \). Let \( N(I) \) be the union of all \( N_j \) at iteration (match-conflict resolution-act cycle) \( I \) so that \( N(I) \) defines the current state of pattern matching. Now, the action part of a rule instantiation \( R_j \) may be associated with a set of working memory changes defined by \( d, (\prod A) \). In addition, there may be other functions performed in the RHS of \( R_j \), \( Y(R_j, WM) \), e.g. performing calculations.

Based on the notation introduced in this section, the forward-chaining mechanism can be summarized in the system model of Fig. 5. Notice, computationally three stages have been defined: pattern matching, conflict resolution and actions.

3. Analysis of computational costs

In this section, the computational costs of rule-based processing are developed based on the mathematical model established in Section 2. It is assumed that the system is a pure
production system so that all computations can be seen as stemming from some rule execution.

3.1 Cost of firing a rule

Define the cost of firing an instantiation of rule $R_i$ as $C_i(R_i^*)$. The asterisk denotes $R_i$ as the selected rule of conflict resolution (see Fig. 5). Note, that the cost of firing different instantiations of the same rule may vary. Let the cost of conflict resolution be given by $C_2(N(I))$ where $N(I)$ is the state of the network as defined earlier. Also, suppose $d_i(\prod A)$ represents the set of WMEs created as a result of firing $R_i^*$. (Removing a WME is analogous to creating a WME, and thus, will not be explicitly discussed). Define $C_4(w_i; N_k)$ as the cost of matching WME, at node $N_k$, the updated WME, at node $N_k$ in the discrimination network and $C_{sk}(w_i)$, which is assumed to be independent of the current state of WM, as the cost of testing attribute $k$ on $w_i$. $K_i$ is the set of attribute tests for the WME class corresponding to WM element $i$. This $K_i$ is dependent on the network structure, in that, it is possible for some attributes to be tested more than once and others to not be tested at all. Then the cost of matching $w_i$ is:

$$C_i(w_i; N) = \sum_{N_k \in N(I)} C_4(w_i; N_k) + \sum_{k \in K_i} C_{sk}(w_i) + \text{Overhead}, \quad (6)$$

where

$$C_4(w_i; N_k) = \sum_{w_j \in N_k} c_4(w_i, w_j),$$

c_4 is the cost of matching WMEs $w_i$ and $w_j$ at node $N_k$, and $C_4(w_i; N_k) = 0$ if $w_i$ does not encounter $N_k$. Normally, the cost of matching WMEs or testing attributes will not vary significantly with their respective values or with the particular matching operation. In this case, $C_4(w_i; N_k) \propto [N_k]$ and $\sum C_{sk}(w_i) \approx |K_i|$ where $|\cdot|$ denotes set cardinality. Now, allow $C_6(Y(R_i^*, WM))$ to be the cost of executing the actions of $R_i^*$, $Y(R_i^*, WM)$. Then $C_i(R_i^*)$, the cost of firing rule instantiation $R_i^*$, is:

$$C_i(R_i^*) = C_2(N(I)) + \sum_{w_i \in d_i(\prod A)} C_3(w_i; N) + C_6(Y(R_i^*, WM)) + \text{Overhead}. \quad (7)$$
Here, the cost of firing a rule has been expressed in terms of the network condition and the effects of firing that rule on the network. This expression treats the computations at negative nodes the same as other nodes. Furthermore, the pattern matching costs are assigned to the effects of the firing of a rule not to the pattern matching required to satisfy that rule. The cost of firing a rule is dependent on the entire rule base as represented by the discrimination net. So, for illustration, one could add a rule whose execution requires very little computation time but whose pattern matching requirements cause other rule firings to be slow.

3.2 Cost of firing a chain of rules

Problem-solving proceeds by successively applying rules to WM. Each rule execution represents a step in the line of reasoning followed to reach a final conclusion. It is possible to define relationships between rules based on their conditions and actions so that, a search tree, for example, would be constructed to indicate all possible lines of reasoning. Two rules have a cause-effect relation if the changes in WM from $R_i$ can cause $R_j$ to be satisfied. That is:

$$d_i(A) \Rightarrow f_j(A_j) \in \Omega_j$$

for some scenario (value of domain attributes). Note, the set of all such rule pairs may be quite large. Thus, to construct a search tree of all possible decision paths would be a very difficult task. If such a tree can be built, however, the worst case computation time can be obtained by finding the most time-consuming rule-chain. Rather than finding all these lines of reasoning and enumerating costs along each path, an unordered set of rule firings will be considered. Let $R$ be this set of rule firings, i.e. $R = \cup R^*_j$, and $C$ be the total cost of computation, then:

$$C = \sum_{R^*_j \in R} C_1(R^*_j).$$ (8)

4. Bounding computational costs

In placing an upper bound on computational time, the time used will be separated into the worst case time for pattern matching and conflict resolution at each step, and the line of reasoning which causes worst case costs to be chained together. For pattern matching, computations become significant when large numbers of intercondition tests, corresponding to relating different elements in WM, must be performed, i.e. when $|N_k|$ is large. In order to find the computation time, the following must be known about WM and the rule structure: the number of WMEs which satisfy rule conditions at node $k$, $|N_k|$, and the subnetwork which defines the relationships between the various WME classes, i.e. the path followed in the net based on the rule conditions.

The cost of conflict resolution depends on the number of rules in the conflict set and the scheme used for rule selection. In this study, it will be assumed that the cost of conflict resolution is independent of the specific criteria which differentiates any rule instantiations; this is consistent with our observations. As a result, the conflict resolution time is a linear function of the size of the conflict set and the worst case occurs when the conflict set is largest.

Finding the worst case line of reasoning is difficult. For instance, the scenario that causes the longest chain of reasoning (most rule firings) does not necessarily result in the slowest
response. Also as mentioned, with the exception of very simple applications, it is impractical to exhaustively search all lines of reasoning to determine the worst case. To obtain an upper bound, it is proposed to analyze the WM changes that lead to increased computations. Each possible chain of reasoning does not need to be determined, instead the worst case contributions based on WM changes from individual rules can be combined without ordering the rule firings. The proposed algorithm analyzes rule relationships by subdividing the rule-base and explicitly defining relationships between WM elements. Notice, the problem of bounding the computation time has changed from one of finding the line of reasoning to one of finding states of the network \( N \). These ideas are formalized in Sections 4.1 and 4.2.

### 4.1 Bounding the rule firing cost

The condition part of a rule can contain a number of attributes, which are organized into a net (Fig. 4). In the worst case, all attributes of a WME will be tested during pattern matching. This corresponds to the case when a WME, \( w_i \), is in all nodes at level \( d = 1 \) in \( N \). Let \( \tilde{C}_s \) be this cost so that:

\[
\tilde{C}_s(w_i) = \sum_{k \in K_i} C_{s_k}(w_i).
\]

(9)

The number of intercondition tests at a node \( N_k \) equals the product of the number of WMEs from each incoming node, so to determine the tests performed for a WME \( w_i \), the number of elements at each node as well as the nodes encountered by \( w_i \) is needed. Define the bounds:

\[
\tilde{N}_k = \max_{N_k} |N_k|,
\]

(10)

\[
\tilde{N} = \bigcup_k \tilde{N}_k.
\]

(11)

Now, a bound on the cost of intercondition testing is:

\[
\tilde{C}_s(w_i) = \sum_{N_i \in \tilde{N}} C_4(w_i; N_k).
\]

(12)

So for the worst case cost of matching an element \( w_i \):

\[
\tilde{C}_3(w_i) = \tilde{C}_s(w_i) + \tilde{C}_3(w_i) + \text{Overhead}.
\]

(13)

We obtain:

\[
\tilde{C}_3(w_i) \geq C_3(w_i; N).
\]

(14)

Observe that the terminal node (rule node) is just a special case in the network, so that, the maximum number of rule instantiations, and thus, the maximum size of the conflict set, occurs under \( \tilde{N} \). That is, the cost of conflict resolution is bounded, \( \tilde{C}_2(\tilde{N}) \geq C_2(\tilde{N}I) \). Furthermore, allow \( C_6(Y(R_j)) \) to be the maximum over \( \Omega \). Combining the above with (8) it follows that if:

\[
\tilde{C}_1(R_j) = \tilde{C}_2(\tilde{N}) + \sum_{w_j \in d(R_j)} \tilde{C}_3(w_i) + \tilde{C}_6(Y(R_j)) + \text{Overhead}.
\]

(15)
Then
\[ C_1(R_j) \geq C_1(R_j). \] (16)

In the preceding development, no assumptions were made on the time to test a particular condition. Normally, this time is relatively independent of the actual test or can be easily bounded. If we assume testing is independent of the value of \( w_i \), then the following simplifications can be written:
\[ C_5(w_i) = |K_1|c_5, \] (17)
\[ C_4(w_i) = \sum_{N_k \in \tilde{N}} |\tilde{N}_k|c_4, \] (18)
\[ C_3(w_i) = \sum_{N_k \in \tilde{N}} |\tilde{N}_k|c_4 + |K_1|c_3 + \text{Overhead}. \] (19)

4.2 Bounding the rule chaining cost

Recall that \( R \) is the set of all rule firings. The worst case is not simply the inclusion of the most computationally expensive rules, but, the set of rules which in combination lead to the slowest response time. Thus, the interaction between different rules must be considered. This interaction is not easy to determine. In the following, the worst case set of rule firings will be determined by analyzing the number of possible executions for a given rule (or rule sets) and selecting only the slowest rule among mutually exclusive rules.

Now, let \( \tilde{R} \) be the worst case set of rule firings. If an instantiation of \( R_i \in R \) for any \( \Omega_i \), then \( R_i, R_j \in \tilde{R} \) unless \( R_i, R_j \) are exclusive, i.e. \( R_i, R_j \in R \not\Rightarrow R_i, R_j \not\in \tilde{R} \), equivalently, \( f_j^{-1}(\Omega_i) \cap f_j^{-1}(\Omega_j) = \emptyset \), in that case, if \( \tilde{C}(R_i) \leq \tilde{C}(R_j) \) only \( R_j \in \tilde{R} \). Note, the previous is easily generalized to multiple rules which allows for cause-effect relationships. In related work an algorithm has been developed for this computation [13]. So, the computational cost \( C \) is bounded by:
\[ C \leq \sum_{R_j \in \tilde{R}} \tilde{C}_i(R_j). \] (20)

5. Practical considerations of upper bound

Computation and application of the derived bound is presented in this section. The emphasis here is how to determine reasonable values for the subcomponents which form Eq. (20).

5.1 Finding worst case components

By examining the possible pattern matching, the conflict set and the characteristics of rule firings, an upper bound can be placed on the computation time. This approach required determination of several quantities: the number of possible attribute tests for a given WME, \( |K_i| \), the limits on the numbers of WMEs which satisfy various conditions in a rule, \( |N_k| \), the limits on the cost of conflict resolution, \( C_3(N) \), and a set of rule firings, \( R \). Based on the developed system model, these quantities are defined by the inverse mapping \( f_j^{-1} \) from the problem space \( \Omega_i \) to the attributes \( A \). This mapping is implicitly defined by the programmer. The difficulty in determining a reasonable upper bound is directly related to the organization
of the rule-base. This is consistent with reports on the importance of highly structured rule-based systems [3, 23] to assist development and analysis.

The worst case pattern matching occurs when the most possible elements are satisfied at each node in the network. To determine the number of elements which satisfy any individual conditions within a rule one must look at characteristics of the problem domain. For instance, if a rule condition is related to a certain class of physical devices, then the maximal set of elements is simply the number of physical devices. Some WM classes may be derived indirectly from the physical system but the number of elements is usually still easy to determine. More involved is the computation of the number of WME combinations which satisfy multiple rule conditions, i.e., deeper nodes in the discrimination network, since interrelationships between WM classes must be determined. A bound can always be obtained from the product of the incoming node sizes. Often this may be improved by simple observation of the domain characteristics. A simulation approach to determining these quantities has arisen from using Petri Nets to model rule-based systems and performing reachability analysis [19, 25]. Practically, computational time for creating $w_i$ can be measured by simply running the ES with a constructed maximal network.

The above procedure establishes $C_i$ and $C_3$. The worstcase cost of pattern matching is obtained by combining all worstcase costs of working memory changes in $d_i$. Also, notice that the conflict set is the largest possible since the maximum number of instantiations occurs for each rule. Furthermore, it is assumed $C_i(Y(R_i, WM))$ is known. Thus, the worst case cost of firing any rule instantiation is obtained.

In order to complete the algorithm, $R$ must be determined. This process can be aided by a rule-base which is broken down into groups or tasks. Typically, these tasks form a natural partitioning of the rule-base and the worst case for this smaller rule-base is determined. For each individual task, we can allow the worst case for all rules within that task. The number of rule firings is found by examining which rules within a task exclude other rules from firing and cause-effect relationships [13]. For instance, consider a task that deals with various failure modes of system components. For a particular component, say, only one type of failure is possible, and thus, only the rule for that particular failure type can be included in the worst case rule firings for the component. The number of executions of the task is determined in a similar manner. In the following, the approach for determining the worst case quantities is summarized.

**Computational bound algorithm**

1. Determine maximum WMEs for each WM class. Joint nodes at a deep level within the network can be the product of incoming nodes or if more detailed information is known this can be used. This establishes $N_a$.
2. Select a task. This is easily done if the rule-base is modular so that any module can be selected. Otherwise, rules should be grouped based on the amount of mutual interaction in order to simplify determination of cause-effect relations and exclusiveness.
3. Select a rule instantiation within the chosen task. The same procedure used to obtain the state of WM in (1) from above can be used to find the worst case pattern matching, $C_4$, for each WME change within $d_i$ and conflict resolution, $C_2$. Here, the maximum cost of $Y(R_i, W_m)$ is assumed known.
4. Compute $C_i$ for each instantiation.
5. Repeat steps 2–4 for each rule within the selected task.
6. Add the computation time of a rule instantiation to the worst case cost only if it is more expensive than all rules with which the rule is exclusive [13], i.e., form $R$.
7. Repeat steps 2–6 for each task and possible multiple executions of a task.'
5.2 Quality of the proposed bound

The following factors affect the tightness of the derived bound:
(1) The physical system is not completely modeled within the ES so that the computed worst case may not be possible. If expert systems possess incomplete system models, further system description is necessary for analysis [25].
(2) Attributes may not be independent. $|N_k|$ was determined by allowing maximum intercondition testing to take place under the creation of any WME. However, dependent attributes may decrease the possible WMEs, and therefore, the computed $|N_k|$ will be higher than the true value.
(3) For practical implementation, rule exclusions are computed within tasks. This, in essence, assumes worst case scenarios in different tasks will occur together, which may not be true.
(4) Conflict resolution depends on the resolution scheme. Conflict resolution computations are normally fewer than pattern matching and the dependence small. Hence, the effect should be negligible.

5.3 Improving computational performance

Optimization of computational performance is difficult due to the complexity and non-procedural nature of the rule-base. In [2, 9, 23, 25], some suggestions for improving efficiency are given. The upper bound derived in this paper is intended as a performance index to guide improvements. The individual contributions to computational cost, that is, $C_2$ the conflict resolution operations, $C_4$ the cost of intercondition testing and $C_5$ the cost of attribute testing associated with the set of rule firings $R$, can indicate where improvements are most needed. In general, determination of computation time is complex. However, based on the analysis in this paper, over simplified efficiency evaluation, such as mere counting of rules or rule firing rates, is seen to be misleading.

6. Numerical examples

In this section, the proposed methodology is demonstrated on an ES developed for electric power system operations. A brief description of this system follows:

VCES (Voltage Control Expert System) is concerned with the reactive power/voltage control [11, 12]. The rule base of VCES incorporates empirical rules for selecting controls, e.g., shunt capacitors, transformer tap changers, generator excitation, in order to resolve violations of load substation voltage limits.

In all examples, VCES is controlling the IEEE 30-bus power system (a standard power system model) with some minor modifications [11]. It should be noted that this problem domain contains an infinite number of possible power system voltage profiles, hence the true worst case can not be found. To begin, a task composed of 9 rules is chosen. This task applies a few simple rules to determine the effect of a control action on neighboring buses (substations). Although only one part of the problem solving process, this task is representative of the increased computation associated with more widespread voltage problems (corresponding to more WME matchings).

Figure 6(a) shows a representative structure for the LHS of the rules in this task. We can easily establish $N_d$ for $d = 1$, corresponding to the 7 WM classes shown for this rule. Our sample system has 14 reactive power controllers (14 WMEs can satisfy 3), there are 30 buses
Finally, possible information voltage each of information impossible to control (12) time (24, 20).

This results correspond to equations (10)–(12) and are sufficient to calculate (16). This rule only fires once as the effect of a particular control on a particular bus is only computed once.

This is repeated for all WM classes and for the rules within this task. Computation of $C_4$ is based on the WMEs created by firings of the selected rules. For the example rule of Fig. 6(a), the WMEs are from the ‘voltage_violator’ WM class and the ‘bus’ WM class. So, the possible contribution to (20) from this rule is known and similarly for the other rules. Finally, the worst case set of rule firings is determined. Note, individual tasks are considered
to simplify determination of rule relationships, however, the entire network must be
considered when calculating $C_4$. Now, the upper bound (20) can be calculated.

The calculated bound for this task is shown in Table 1. Notice, that the number of
executions grows with the number of controls (14 in the example power system). So, one
expects the worst case time to increase with the number of controls selected in two ways:
increased pattern matching and increased executions of the task. Table 2 shows the
breakdown of costs in terms of pattern matching, conflict resolution and the actions taken in
rule firings including overhead for the worst case single execution of the control effect task.
It can be seen that the dominant cost (88%) for this task is pattern matching. Using this as a
guide, a number of different rule implementations were developed by varying the data
structure and the ordering of conditions in the rules (Fig. 6).

The first two modified rule structures, 6(b)–(c), reorder conditions in order to investigate
the effect of a different rule ordering. Further, all rules in the task were modified in the same
manner so that sharing of conditions could be maintained (at least within this task). The
third structure combined several WME classes into a single class eliminating the need for
repeated pattern matching between these WME classes. Again, rules were written to 'maximize'
condition sharing. This new rule structure also affects the rule firings during the
overall solution process, e.g. some data is obtained in a single rule firing rather than
repeated rule firings. As a result, computational savings can be expected in the rule chaining
for the overall system performance.

For each of the rule structures indicated, the upper bound on computations was de-
determined. These bounds were compared to the computation time spent in an example with
two low voltage substations which is typical among scenarios investigated here. From the
results of Table 3, it can be seen that inappropriate ordering of conditions can create
excessive matching, leading to very slow response (structure 1 versus structure 2). The major
improvement in computation for structure 2 comes from testing conditions 1, 3, 6 and 7 first
as only 1 WME combination can arise from these conditions. Also, Table 3 indicates that a
carefully designed data structure can decrease matching significantly; structure 3 is about 30
times faster than structure 1 based on the upper bounds. Structure 3 decreases PM

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst case bound for control effect task in VCES</td>
</tr>
<tr>
<td># of executions</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

(s-seconds)

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakdown of computational costs for upper bound on control effect task</td>
</tr>
<tr>
<td>Operation</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>Actions ($C_4$)</td>
</tr>
<tr>
<td>Conflict resolution ($C_2$)</td>
</tr>
<tr>
<td>Pattern matching ($C_3$)</td>
</tr>
<tr>
<td>Total (C)</td>
</tr>
</tbody>
</table>

(ms–milliseCONDS)
Table 3
Measurements on rule structures for control effect task

<table>
<thead>
<tr>
<th>Rule structure</th>
<th>Upper bound (Eq. 20)</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>2.54 s</td>
<td>0.08 s</td>
</tr>
<tr>
<td>Structure 1</td>
<td>36.25 s</td>
<td>0.35 s</td>
</tr>
<tr>
<td>Structure 2</td>
<td>1.42 s</td>
<td>0.06 s</td>
</tr>
<tr>
<td>Structure 3</td>
<td>1.13 s</td>
<td>0.05 s</td>
</tr>
</tbody>
</table>

Table 4
Overall performance comparison for VCES

<table>
<thead>
<tr>
<th>Rule structure</th>
<th>Worst case found</th>
<th>Upper bound (Eq. 20)</th>
<th>Averaged (24 cases)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>55.8 s</td>
<td>67.4 s</td>
<td>4.09 s</td>
</tr>
<tr>
<td>Modified rule base</td>
<td>43.1 s</td>
<td>54.8 s</td>
<td>3.50 s</td>
</tr>
</tbody>
</table>

% Time saved 22% 19% 14%

computations partly from obtaining a list of controllers before a list of load substations (there are far fewer controls than load substations in general). Additionally, combining WM classes eliminated searching for some sensitivity factors to determine the resulting voltage correction amount. Variation in rule chaining arising from structure 3 can not be seen in this task.

Measurements were made on the entire rule-base of VCES using the original rule structure and a modified rule structure which combined some WME classes and reordered conditions in a similar fashion to Fig. 6(d). Worst case response improved by 22% as a result of these modifications (Table 4). Since the true worst case is impossible to find (infinitely many scanarios), the computation time of a number of identified slow examples was measured and the worst over these was used as the ‘worst case found’ in Table 4. Note in Tables 3 and 4, that the average case improved when the worst case cost decreased. Hence for this system, the worst case scanario provides some insight into the formation of data and rules for other scenarios as well. However, even without this insight worst case computation time is an important characteristic for real-time performance.

Finally, in VCES, there is significant computational savings associated with maintaining most of the power system data as a database. Relevant data is retrieved using standard database techniques. This saving although important, is not analysed here.

7. Conclusion

This paper proposes a new method to analyze the computational costs of rule-based systems. The chaining mechanism is modeled as a system which processes WMEs through a number of tasks. By analyzing the functions of each task, an upper bound is derived for the computation time. In general, analyzing the behavior of a rule-based system is difficult, since rules are selected based on the specific scenario. By considering the worst case situation, this study achieves an explicit performance index for rule-based system design and evaluation. The numerical examples in this paper show that very similar rule-base implementations can have significantly different computational efficiency. This indicates that careful analysis is important for any rule-based system, such as VCES, for which fast response is required.
Similarly, for any approach to improving computational performance, it is important to establish a foundation for analyzing the improvements. This research is a step in that direction.

Acknowledgements

This work was supported in part by the National Science Foundation through grant No. ECS-8657671.

References

Kevin Tomsovic received the B.S. from Michigan Tech. University, Houghton, in 1982, and the M.S. and Ph.D. degrees from University of Washington, Seattle, in 1984 and 1987, respectively, all in Electrical Engineering. He has held positions at McDonnell Douglas, IBM, and MIT Lincoln Laboratory. Visiting university positions have included Boston University, National Cheng Kung University, National Sun Yat-Sen University and the Royal Institute of Technology in Stockholm. Currently, he is an assistant professor at Washington State University. His research interests include expert system and fuzzy set applications to power system control and security.

Chen-Ching Liu received his BSEE and MSEE degrees from National Taiwan University, Taiwan, in 1976 and 1978, respectively. After 2 years of service at the Army Signals and Electronics School, Taiwan, he joined the University of California, Berkeley, and completed his Ph.D. in EECS in 1983. He is presently Professor of Electrical Engineering at the University of Washington, Seattle. Dr. Liu received a Presidential Young Investigator’s Award from the National Science Foundation, USA, in 1987. He served as Guest Editor of the special issue, Knowledge-based systems in electric power systems, Proceedings of the IEEE, May 1992. Dr. Liu held the visiting positions of Tokyo Electric Power Chair at the University of Tokyo, Japan, in Spring 1991 and Professeur Invite at the Swiss Federal Institute of Technology, Lausanne, in Summer 1989 and 1992.