A Reliability Model for Real-Time Rule-Based Expert Systems

Ing-Ray Chen, Member IEEE
National Cheng Kung University, Tainan
Tawei Tsao
University of Mississippi, University

Key Words — **Rule-based expert system, Reliability model, Real-time, Simulation, Petri net, Markov process.**

Reader Aids —
General purpose: Present a reliability model of expert systems
Special math needed for explanations: Petri net concepts
Special math needed to use results: None
Results useful to: Expert-system designers and reliability analysts

Summary & Conclusions — **This paper uses two modeling tools to analyze the reliability of real-time expert systems: 1) a stochastic Petri net (SPN) for computing the conditional response time distribution given that a fixed number of expert system Match-Select-Act cycles are executed, and 2) a simulation search tree for computing the distribution of expert system Match-Select-Act cycles for formulating a control strategy in response to external events. By modeling the intrinsic Match-Select-Act cycle of expert systems and associating rewards rates with markings of the SPN, the response time distribution for the expert system to reach a decision can be computed as a function of design parameters, thereby facilitating the assessment of reliability of expert systems in the presence of real-time constraints. The utility of the reliability model is illustrated with an expert system characterized by a set of design conditions under a real-time constraint. This reliability model allows the system designers to: 1) experiment with a range of selected parameter values, and 2) observe their effects on system reliability.**

1. INTRODUCTION

Acronyms & Abbreviations

A*	a particular heuristic-based, *conflict-resolution* algorithm [1, 27]
AI	artificial intelligence
SPN	stochastic Petri Net.

An embedded rule-based expert system is characterized by its intrinsic Match-Select-Act [29] execution cycle in which it responds to an external event (*eg*, a sensor event which inputs *facts*) by first matching arriving facts against the l.h.s. condition elements of the rules comprising the expert system (match phase), then selecting a rule to fire among the rules that are instantiated (select phase), and finally executing the r.h.s. actions of the selected rule (act phase). Firing a rule can generate more new facts, causing the Match-Select-Act cycle to activate again. This process continues until some newly generated facts meet the termination condition, at which point the expert system has reached a decision and the sequence of rules fired on the solution path is the formulated strategy in response to the external event.

Previous studies regarding the Match-Select-Act cycle have centered on the performance analysis of each separate phase. Furthermore, these studies are conducted independently.

For the match phase, the emphasis is on performance improvement due to using certain matching algorithms [21, 24, 28] such as the Rete [11] and Treat [20] algorithms. The basic principle behind these matching algorithms is that, instead of matching all the facts in the working memory against the r.h.s. condition elements of all the rules in the rule set, only the rules that are actually affected by newly generated facts in each cycle are evaluated, thereby speeding-up the match phase per cycle. Expert system compilers based on these matching algorithms have been implemented for OPS5 production system programs [10, 15, 21]. These studies: 1) focused on the (sequential or parallel) speed-up of the match phase [21] without considering the select phase, or 2) simply assumed that a static conflict-resolution policy has been used (*eg*, the salient feature in CLIPS [12] or the top-rule first policy [14]).

For the select phase, one research area is aimed at reducing the number of expert system cycles for reaching a decision by using a heuristic-based algorithm to resolve conflicts regarding which rule or group of rules should be fired, typically by modifying an existing conflict-resolution strategy of an expert system shell such as OPS5 [6, 17] or AF [8, 13]. A related research area focuses on obtaining the average number of nodes expanded (and thus the average number of cycles executed) by a heuristic-based, *conflict-resolution* algorithm such as A* [1, 27], assuming that the time required to expand a node in an expert system inference tree or graph is constant. The assumption that 'the cycle time less the time for the select phase' (time for matching & acting) is constant is not appropriate for expert systems. Also, the average time-complexity analysis is not useful for system reliability assessment because it cannot account for the variation of the system response time.

For the act phase, the performance gain due to parallel rule firing has been investigated [15, 16]. The basic idea is to reduce the number of Match-Select-Act cycles by applying an interference detection algorithm in the select phase to choose as many non-interfering rule instantiations as possible [16]. These non-interfering rule instantiations are then fired simultaneously in the act phase. Since all non-interfering rules can be fired in every cycle, assuming infinite processing elements are available, there is no need to use any conflict-resolution algorithm in the select phase. These studies (*eg*, [16]) do not include the effect of matching. Furthermore, similar to the studies conducted in other phases, the performance evaluation concentrates only on the average time behavior of the system.

We propose a model that ties in all three phases of the Match-Select-Act cycle. Unlike previous studies, the objective is not to devise new matching, conflict-resolution, or firing algorithms for improving the performance of individual phases; rather, we are interested in providing a framework with which the system reliability of real-time expert systems can be analyzed.

0018-9529/95/$4.00

In doing so, an equation for system reliability of expert systems is first derived. Then, by using a reward SPN [18] describing the intrinsic Match-Select-Act cycle of expert systems to account for the response time variation in all three phases, the parameters identified by the equation are parameterized (*ie*, computed using analytic or statistical means), allowing the system reliability to be computed.

2. DEFINITIONS & ASSUMPTIONS

Notation

R_{system} system reliability
t_p, t_e [planning, execution] time
l_i reliability of hardware component i during time t_p+t_e
$\boldsymbol{l}$ $\{l_1,\ldots,l_n\}$
λ_s constant software-failure rate due to software residual faults
$R_{h,\text{gen}}(\tau;\ \boldsymbol{\phi})$ general hardware-reliability function; τ, $\boldsymbol{\phi}$: various [times, parameters]
$P_C(t_p)$ Pr{heuristics used in the planning method work during time t_p}
$P_V(t_p,t_e)$ Pr{plan is valid during time t_p+t_e|plan is based on observation at time 0}
$f(t)$, $F(t)$ pdf{t}, Cdf{t}
t_R real-time constraint for a problem-solving request
m_{rtf} average number of fact changes per rule firing
m_{ftr} average number of rule instantiations per fact change
$f(t_p|j)$ pdf{response time|j Match-Select-Act cycles are executed to reach a decision}
$f_{\text{cyc}}(j)$ pdf{number of Match-Select-Act cycles to reach a decision}
$f_{\text{cyc}}(j|N)$ pdf{number of Match-Select-Act cycles to reach a decision|solution is found at depth N of the search tree}
b_p, b_n average number of [positive, negative] rule instantiations per rule firing
b $b_p - b_n$
P_{addr}, P_{remover} Pr{when a rule fires, a new rule instantiation generated is a [positive, negative] rule instantiation and is [added into, removed from] the conflict set}
$g(n)$ cost of the path accumulated from the root node to node n in the search tree
$h^*(n)$ remaining cost from node n to a nearest goal node in the search tree
$h(n)$ an estimate of $h^*(n)$ such that $0 \le h(n) \le h^*(n)$
$Y(n)$ $(h^*(n) - h(n))/h^*(n)$: error of the heuristic estimate on node n
ϵ error bound on $Y(n)$ for each node n in the search tree
$N_{\max}$ maximum depth of the search tree
N depth of the search tree at which a solution is found
$P(N)$ Pr{a solution is found at depth N of the search tree}; $\Sigma_{N=1}^{N_{\max}} P(N) \equiv 1$
μ_t transition rate of transition t in the Petri net
R number of rule instantiations in the conflict set
T average time to scan a node in a sorted linear list with each node corresponding to a rule instantiation stored in the list
T_h average time needed to compute the heuristic estimate $g(n)+h(n)$ for a rule instantiation n
T_a average time needed to execute a r.h.s. action
Z number of nodes (rule instantiations) expanded by A* in a b-ary tree
(α, β) range of arc costs in the search tree
$\mathfrak{I}(\cdot)$ indicator function: $\mathfrak{I}(\text{True})=1$, $\mathfrak{I}(\text{False})=0$.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

2.1 Definitions

The system reliability of an expert system can be defined as the probability that, for a problem-solving request (or mission) issued from the environment, the system can successfully plan and execute a response without causing a hardware, software or timing failure [7]:

$$\Pr\{\text{mission succeeds}\} = \int_0^\infty \int_0^\infty R_{h,\text{gen}}(t_p,t_e;\ \boldsymbol{l}) \cdot \exp(-\lambda_s \cdot t_p) \cdot P_C(t_p) \cdot P_V\ (t_p,t_e)\ dF(t_p,t_e). \quad (1)$$

This reliability definition is on a per problem-solving request basis. Problem solving requests are s-independent from each other and are issued from the environment as demanded by the occurrences of real-time events. We consider hard real-time environments where a violation of real-time deadline is a system failure. The notion of "soft" timing failures where the accumulation of a series of "soft" timing failures can cause a system failure [4] is not considered in this paper.

Two types of faults are considered in (1).

1. Type #1 is due to software residual faults (program bugs) which, when executed under particular conditions, can cause a software failure. For this type of fault, existing software reliability models [23] can be used for estimating the software-failure rate. In general, this software-failure rate decreases as more time is spent in testing & debugging the program. Since typically the program is not modified in the operational phase, the software-failure rate due to program bugs can be considered constant in the operational phase: λ_s in (1) is modeled as a constant. ◀

2. Type #2 is due to the fundamental AI techniques used by the expert system program which might not work all the time. For example, a conflict-resolution algorithm used by an expert system program might find only sub-optimal solutions and might even fail to find a solution in certain cases, especially when there is a stringent real-time constraint. This type of fault exists even if the program is devoid of program bugs (such as a bug in implementing the algorithm, or constructing the rule set); and it is not removable even though the system has been tested & debugged for a long time. This fault type can cause the following failures to occur.

a. *Fuzzy-output failures* [3, 4]: The output solution found by an expert system program might not be correct all the time. The degree to which the output is acceptable can be modeled

by Pr{search heuristics work in t_p}, *ie*, $P_C(t_p)$ in (1) with the value of 1 representing that the output is totally acceptable, and 0 representing that the output is totally unacceptable. A strong *s*-correlation exists in expert system programs between t_p & $P_C(t_p)$.

b. *Timing failures:* In trading response-time with program correctness, if the expert system chooses to compromise the system response time in searching for a more acceptance output in a combinatorial search space, it can violate the real-time deadline requirement.

c. *Hardware failures:* If the expert system chooses to compromise the program correctness in searching for a quicker but inferior solution in order to avoid timing failures, the inferior solution might use inferior hardware components in executing the solution and can cause a hardware failure with a higher probability. This effect is modeled by $R_{h,\text{gen}}(t_p,t_e;\boldsymbol{l})$ in (1).

2.2 Assumptions

1. The real-time expert system runs under a hard real-time environment so that the system fails if it fails to make & execute a decision within t_R:

$$P_V(t_p,t_e) = \mathcal{G}(t_p+t_e \le t_R).$$

2. The expert system runs on a hardware platform that obeys the exponential failure law with a constant λ_h:

$$R_{h,\text{gen}}(t_p,t_e;\boldsymbol{l}) = \exp(-\lambda_h \cdot (t_p+t_e)).$$

3. The execution time of the formulated strategy is immediate, *ie*, the operator can immediately execute the strategy once it is formulated by the expert system:

$$f_e(t_e) = \delta(0),$$

$\delta(t) \equiv$ standard impulse function having unit area concentrated in the immediate vicinity of t. [This assumption removes the possibility of hardware failures in the execution phase.]

4. There is no software residual fault, *ie*, the expert system has been tested for a long time before release so that $\lambda_s \approx 0$. [This assumption removes the possibility of software failures caused by program bugs.] ◄

Under these assumptions,

$$\Pr\{\text{mission succeeds}\} = \int_0^{t_R} \exp(-\lambda_h \cdot t_p) \cdot P_C(t_p)\ dF(t_p). \tag{2}$$

These assumptions are reasonable for expert systems. Expert systems typically:

- provide control strategies only to human operators whose execution speed can be considered immediate.
- run on microprocessor-based computers whose hardware-failure rate will be considered constant.

Eq (2) implies that the most important factor affecting system reliability is $F(t_p)$. This makes intuitive sense because the high variability of the response time of expert systems is a major reason why expert systems have been used for advisory rather than for control purposes. Section 3 uses a reward SPN model to compute $F(t_p)$.

Other than $F(t_p)$, eq (2) also requires the *parameterization* of λ_h & $P_C(t_p)$. While the assessment of λ_h is subject to standard techniques [2], $P_C(t_p)$ depends on whether the conflict-resolution algorithm used in the select phase has any fundamental limitations. For example, the use of heuristics in a conflict-resolution algorithm can result in occasional failures even though the algorithm is devoid of any software residual faults (even $\lambda_s = 0$). For exhaustive-search-based (such as depth-first and breadth-first) conflict-resolution algorithms, $P_C(t_p) = 1$ because the probability that the search can be misled by heuristics is zero since no heuristics are ever used. However, this might result in performance degradation because rules are selected statically (*eg*, the static priority policy in CLIPS [12]) without considering the possibility that selecting rules heuristically might lead the system to reach a decision more quickly. The performance of the system can be improved by using heuristic-based conflict-resolution algorithms in the select phase (such as the Means-Ends-Analysis algorithm in OPS5 based on recency & specificity [19] or A*). However, to avoid compromising system reliability, some admissibility conditions might have to be satisfied to guarantee $P_C(t_p) = 1$. For example, if A* is used as the conflict-resolution algorithm, then the admissibility condition to guarantee $P_C(t_p) = 1$ is that the heuristic estimate of *any* rule instantiation in the conflict set, *ie*, the estimate of the distance in terms of the execution cost from *a* rule in the conflict set to the nearest rule leading to termination, must be not greater than the actual distance of that rule [27]. When this condition is violated, A* can lead to occasional failures.

Assumption

5. The admissibility requirement of the conflict-resolution algorithm is satisfied by the design so that $P_C(t_p) = 1$. However, there exists a variation in heuristic estimates due to the fuzzy nature of expert system processing, thus causing the number of Match-Select-Act cycles executed by the expert system before a decision is reached to be a distribution. ◄

With this additional assumption #5, we remove fuzzy-output failures from our failure model and (2) simplifies to:

$$R_{\text{system}} = \int_0^{t_R} \exp(-\lambda_h \cdot t_p) \cdot f(t_p)\ dt_p. \tag{3}$$

3. THE REWARD SPN MODEL

We use a stochastic Petri net (SPN) to model the Match-Select-Act cycle in expert systems, with the objective of computing $f(t_p)$. The model is shown in figure 1 with circles representing *places* and bars representing *transitions*. The 8 transitions and their representations are:

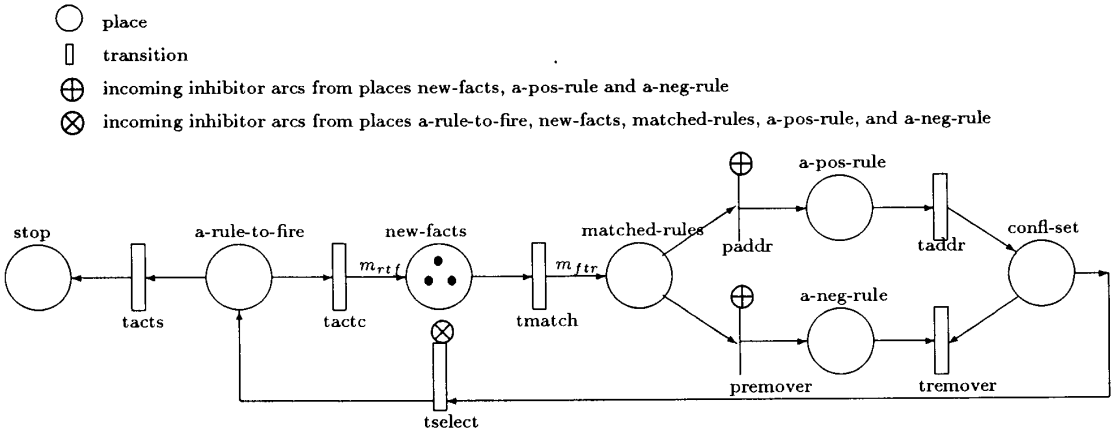

Figure 1. Petri-Net Model for Rule-Based Expert Systems

tactc	act phase leading to the next cycle
tacts	act phase leading to a decision
tmatch	match phase
paddr	Pr{positive instantiation}
premover	Pr{negative instantiation}
taddr	action for adding a positive rule instantiation to the conflict set
tremover	action for removing a negative rule instantiation from the conflict set
tselect	select phase.

The SPN has 7 places which serve as token-holders. Each place can hold an arbitrary number of tokens corresponding to the objects held in the place. For example, the 3 tokens in place new-facts in figure 1 correspond to 3 facts being held in that place, waiting to be matched against the l.h.s. condition elements of rules by the matching algorithm. Similarly, tokens in place matched-rules corresponds to new rule instantiations produced in a cycle, waiting to be added to (with probability P_{add} for positive instantiations) or removed from (with probability P_{remove} for negative instantiations) the conflict set.

The Match-Select-Act cycle is executed by the expert system sequentially in a uniprocessor; this is modeled by adding *inhibitor arcs* from places new-facts, a-pos-rule, and a-neg-rule to transitions paddr and premover as well as from places a-rule-to-fire, new-facts, matched-rules, a-pos-rule, and a-neg-rule to transition tselect. Since SPN enables a transition only when all incoming non-inhibitor arcs emanate from places containing one or more tokens, and all incoming inhibitor arcs emanate from places containing no tokens, the use of inhibitor arcs in SPN ensures that the select phase in a cycle is executed only after the act phase in the previous cycle and the match phase in the same cycle have been executed, even if the conflict set is not empty (a nonempty place confl-set). For example, transition paddr is enabled only if place matched-rules is nonempty and place new-facts is empty. When a transition is enabled, the transition fires immediately, if it is an *immediate* transition, or after an amount of time determined by a random sample from the associated distribution with the transition elapses, if it is a *timed* transition. In our SPN model, only transitions paddr & premover are immediate (a constant 0 distribution), which are associated with probabilities p_{addr} & p_{remover} to model the fact that a new rule instantiation can be a positive instantiation with probability p_{addr}, or a negative one with probability p_{remover}. To simplify the analysis, the firing times of other transitions are exponentially distributed, thus rendering the Petri net stochastic and susceptible to solution techniques provided by the software tool Stochastic Petri Net Package (SPNP) [5, 22]. Our approach can be easily extended to Extended Stochastic Petri Net (ESPN) [9] models in which firing times are general distributions.

Regardless of the distribution type, when a transition is fired, one or more tokens, depending on the *multiplicity*[1] of the associated input arc, are removed from the input place, and one or more tokens, depending on the *multiplicity* of the associated output arc, are added to each output place. The multiplicities of arcs in the SPN model are system parameters and must be modeled explicitly. There are two arcs having multiplicity greater than 1: the output arc from transition tactc to place new-fact (where the arc multiplicity is labeled with m_{rtf} in figure 1); and the output arc from transition tmatch to place matched-rules (where the arc multiplicity is labeled with m_{ftr}). All other arcs only have multiplicity of 1 (the default). m_{rtf} accounts for the fact that firing a rule in a cycle can put one or more facts into place new-facts, depending on the number of actions specified in the r.h.s of the rule selected to fire in the act phase. Since rules are not tagged, the number of facts generated in each cycle is not known *a priori*, but we can associate m_{rtf} with a multiplicity function that generates a random number characterizing the average number of actions in the r.h.s. of all the rules, thus accounting for the stochastic behavior of m_{rtf}. This distribution can be obtained by parsing the expert system rule set [15]. On the other hand, m_{ftr} can be associated with

[1] An arc with *multiplicity k* can be thought of as *k* arcs having the same source & destination.

a multiplicity function that generates a random number characterizing the average number of rule instantiations per fact, which can be obtained by analyzing the run-time characteristics of the expert system [15]. Although the number of rules instantiated per fact change can be determined this way, the matching algorithm handles only one fact change at a time since the arc multiplicity of the input arc from place new-facts to transition tmatch is 1.

A state of the SPN is characterized by the distribution of tokens in the places, called a *marking* of the SPN. Initially, an external (I/O sensor) event puts one or more tokens (depending on whether the sensor is a multiple or single sensing device) into place new-facts to start a Match-Select-Act cycle, thus marking the initial state of the SPN. When a strategy is formulated by the expert system in response to the event, place stop is no longer empty. The termination condition causing place stop to become nonempty is examined by a special enabling function associated with transition tacts. When the specified termination condition is not satisfied, the enabling function disables transition tacts, thus causing transition tactc to fire and, consequently, activating the next Match-Select-Act cycle; otherwise, transition tacts is fired over transition tactc, forcing the expert system to terminate. This is achieved by associating a high priority (1) with transition tacts. By default, transitions have the lowest priority (0) and have no special enabling functions.

Because the probability that the system stays in a particular marking (state) evolves over time as a function of the distributions of transition firing times and the arc multiplicities of the SPN model, the response time distribution of the expert system subject to a specified termination condition, $f(t_p|\text{termination condition})$, can be obtained by associating a reward rate of 1 with those markings in which place stop is nonempty and a reward rate of 0 otherwise. In the context of rule-based expert systems, the termination condition corresponds to the number of Match-Select-Act cycles executed by the expert system to formulate a decision.

$$f(t_p) = \int_0^\infty f_{\text{cyc}}(j) \cdot f(t_p|j)\, dj. \tag{4}$$

The r.h.s. of (4) can be used subsequently to compute the system reliability based on (3).

4. EXPERT-SYSTEM MODEL AND RELIABILITY COMPUTATION

This section illustrates the utility of the SPN model with an expert system model characterized by the design conditions:

1. A rule firing results in an average of b_p positive and b_n negative rule instantiations to the conflict set. (This condition applies to several expert systems [15] where the 'average number of changes made to the conflict set per production rule firing' as well as the 'average size of the conflict set' can be statistically measured.) Further, $b_p > b_n$; therefore the search space for formulating a decision (finding a solution path) is a b-ary tree, meaning that a rule firing, on the average, adds b rules to the conflict set (b nodes to the search tree). ◄

2. A* [1, 27, 29] is used as the conflict-resolution algorithm in the select phase to find a cost-effective solution path. To achieve this goal, A* uses the sum of 2 quantities as a heuristic estimate for the cost of a potential solution path:

   i. $g(n)$. The cost of the path accumulated so far from the root node (the first rule instantiation fired in response to a real-time event) to a candidate rule instantiation n at the front of a partially-explored solution path.
   ii. $h(n)$. The remaining cost from rule n to a nearest rule instantiation whose firing leads to a decision.

A* always selects (and subsequently fires) the frontier rule instantiation of a solution path having the minimum $g(n) + h(n)$. Furthermore, to guarantee $P_C(t_p) = 1$, the admissibility condition $0 \le h(n) \le h^*(n)$ is satisfied, thus guaranteeing that A* always finds a solution path with the cheapest cost [27].◄

3. The error of the heuristic estimates, $h^*(n) - h(n)$, is relative [27] and the error function, $Y(n)$, is uniformly distributed over the interval $[0,\epsilon]$, $0 \le \epsilon \le 1$, with ϵ characterizing the degree to which h(n) deviates from $h^*(n)$ and thus the complexity of the A* algorithm. When $\epsilon = 0$ $(h(n) = h^*(n))$, the number of rules fired is linear in N, the depth of the solution path found by A*; on the other hand, as ϵ increases, $h(n)$ deviates more and more from $h^*(n)$, resulting in the number of rules fired being exponential in N. ◄

4. The conflict set is maintained by a linear list data structure in which all rule instantiations are sorted by increasing $g(n)+h(n)$. In terms of priority ordering, the priority of rule n in the conflict set is inversely proportional to $g(n)+h(n)$, so that the rule with the lowest $g(n)+h(n)$ has the highest priority. Since A* always selects the rule with the highest priority to fire, the rule at the head of the sorted linear list is always selected to fire in every cycle.

5. The last rule instantiation leading to a decision in response to an external event is located at depth N of the search tree with probability $P(N)$, a parameter which can be obtained by analyzing a large sample of event instances. An extreme case is $P(N_{\max})=1$ with all leaf nodes at depth $N_{\max}$ being solution nodes, *eg*, the response trees for nuclear reactor operations [26]. ◄

4.1 Computation of $f(t_p|j)$ Using the SPN model

$f(t_p|j)$ can be numerically computed by parameterizing & running the reward SPN model (figure 1) such that the enabling function associated with transition tacts enables the transition only when j Match-Select-Act cycles are executed by the SPN. Due to the stochastic nature of the SPN model, the processing speed of every Match-Select-Act cycle varies. Therefore, $f(t_p|j)$ depends not only on j, but on the distribution of the transition firing times and the arc multiplicities of the model. For the transition firing times, since we allow all firing times to be either exponentially distributed or immediate, it suffices to parameterize them with their transition rates or

probabilities. Methods for parameterizing the SPN model with the objective of computing $f(t_p|j)$ are now listed.

1. *Transition rate of* taddr (μ_{taddr}). Because the conflict set is maintained by a linear list data structure in which all rule instantiations are sorted by decreasing priority, adding a positive rule instantiation (in place a-pos-rule) to the conflict set (in place confl-set) is a 2-step process:

i. computing $g(n)+h(n)$ for the rule instantiation;
ii. inserting the rule instantiation into the linear list.

Since on the average one half of the list elements are scanned before an insertion can take place, we use:

$$\mu_{\text{taddr}} = (\tfrac{1}{2}R \cdot T + T_h)^{-1};$$

T & T_h can be statistically measured.

2. *Transition rate of* tremover (μ_{tremover}). Removing a negative rule instantiation (in place a-neg-rule) from the conflict set involves first finding the rule instantiation in the linear list and then removing it. Since on the average one half of the list elements are scanned before a deletion operation can take place, we use:

$$\mu_{\text{tremover}} = (\tfrac{1}{2}R \cdot T)^{-1}.$$

3. *Probabilities of* p_{addr} & p_{remover}. A rule firing on the average makes b_p positive and b_n negative rule instantiations to the conflict set; therefore,

$$p_{\text{addr}} = b_p/(b_p+b_n),$$

$$p_{\text{remover}} = 1 - p_{\text{addr}}.$$

4. *Transition rate of* tselect (μ_{tselect}). The rule with the lowest $g(n)+h(n)$ (highest priority) is always at the head of the sorted linear list. Therefore, selecting a rule to fire requires the same amount of time for deleting the head node from the linear list:

$$\mu_{\text{tselect}} = (T)^{-1}.$$

5. *Transition rate of* tmatch (μ_{tmatch}). This is the rate at which the matching algorithm can process a fact change, a parameter which can be determined by analyzing the run-time characteristics of the expert system [15] depending on the specific matching algorithm [11 or 20] used in the match phase. In general, this rate can be obtained by first measuring the average numbers of tests on constant-nodes, α-memory-nodes, β-memory-nodes, and-nodes, and or-nodes per fact change; and then multiplying these numbers with the corresponding times needed for the tests.

6. *Transition rate of* tactc (μ_{tactc}). Because the multiplicity of the output arc from transition tactc to place new-fact is not 1 but is m_{rtf} which varies dynamically, μ_{tactc} must be computed cycle-by-cycle to account properly for the variation. This can be done by associating μ_{tactc} (the service rate of the act phase) with a transition function which generates a random time based on an m_{rtf}-phase Erlang distribution such that the service rate of a single phase of the Erlang function is the same as the rate at which the system can execute a r.h.s action. m_{rtf} represents the number of facts produced (equivalent to the number of r.h.s actions executed) per rule firing and is randomly generated cycle-by-cycle by the multiplicity function associated with the output arc going from transition tactc to place new-fact. Thus, the average service rate of an Erlang phase is $(T_a)^{-1}$; thus μ_{tactc} can be computed as soon as m_{rtf} is generated.

7. *Transition rate of* tacts (μ_{tacts}). Since all the r.h.s actions of the last rule instantiation must be executed by the expert system before a decision can be reached, μ_{tacts} is the same as μ_{tactc}. ◀

Figure 2 shows $f(t_p|j)$ for several j values, after parameterizing & running the SPN model based on the set of parameters in table 1.

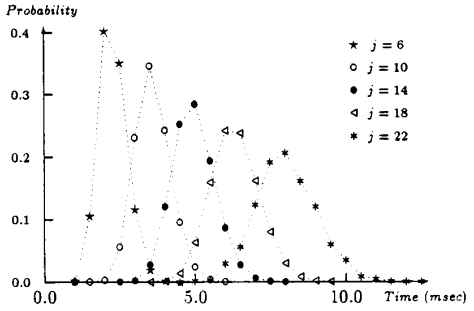

Figure 2. $f(t_p|\text{expert-cycle} = j)$ As a Function of j

Based on (4), $f(t_p|j)$ obtained this way, along with $f_{\text{cyc}}(j)$ (see section 4.2) can be used to compute $f(t_p)$, which subsequently can be used to compute the reliability of real-time expert systems based on (3).

TABLE 1
Parameters for Generating Figure 2

T_h	10 msec
p_{addr}	0.8
p_{remover}	0.2
m_{rtf}	uniformly distributed over (2,4)
m_{ftr}	uniformly distributed over (1,3)
μ_{tmatch}	1/100 msec
T_a	10 msec

4.2 Computation of $f_{cyc}(j)$ Using A Search-Tree Simulation Model

$$f_{cyc}(j) = \sum_{N=1}^{N_{max}} P(N) \cdot f_{cyc}(j|N) \quad (5)$$

The most important factor in $f_{cyc}(j|N)$ is the error of the heuristic estimates, $h^*(n) - h(n)$, for each rule instantiation n in the conflict set. To see this, refer to the equation in [27] for the average number of nodes (rule instantiations) expanded by A* in a b-ary tree, assuming that the depth of the solution path is N:

$$E\{Z\} = N + 1 + (1-b^{-1}) \cdot \sum_{j=1}^{N} \sum_{d=1}^{j} b^d \cdot \prod_{k=1}^{d} q_{j,k}. \quad (6)$$

Notation

$q_{j,k}$ Pr{node $n_{j,k}$, located at depth k of an off-solution-path subtree T_j, is expanded by the underlying search algorithm}

$n_{j,0}$ set of nodes, $0 \le j \le N$ that represents the solution path

Eq (6) covers the best & worst cases.

- Best case, $h(n) = h^*(n)$: for all j,k, $N \ge j \ge 1$, $k \ge 1$, $q_{j,k} = 0$; and $Z_{best} = N+1$;
- Worst case, $h(n) = 0$: for all j,k, $q_{j,k} = 1$.

In these two extreme cases, $f_{cyc}(j|N)$ is the standard impulse function having unit area concentrated in the immediate vicinity of $E\{Z\}$.

However, when $h(n)$ is a r.v. between 0 & $h^*(n)$, no analytic form for $f_{cyc}(j|N)$ exists and statistical or simulation methods must be used.

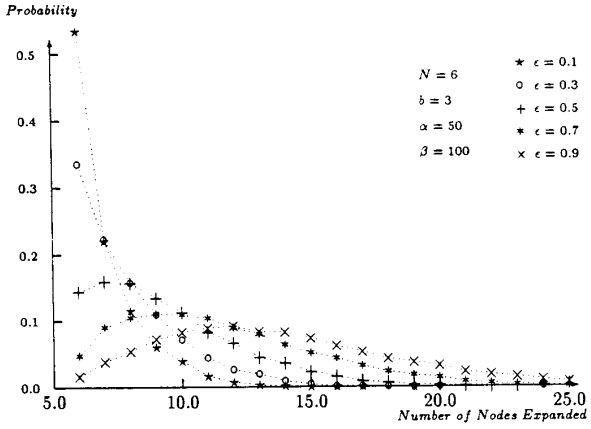

Figure 3. f(Number of nodes expanded|N)

We have conducted simulation studies with the objective of computing $f_{cyc}(j|N)$. In each simulation run, a search tree is constructed in which:

- total depth is N_{max};
- arc costs are uniformly distributed over (α, β), $\alpha \ge \beta$;
- error function $Y(n)$ is uniformly distributed over $(0,\epsilon)$, $0 \le \epsilon \le 1$;
- the solution node (and thus the optimal solution node) that is to be found by A* is located at depth N, but other non-optimal solution nodes might exist at the same or other depths of the tree if that information is given, *eg*, all leaf nodes are solution nodes [26].

A simulation run for computing $f_{cyc}(j|N)$ consists of a sufficient number of replication runs such that the s-confidence level is 95%. In each replication run, N, N_{max}, ϵ, α, β are kept constant, while $Y(n)$ and solution node locations are randomly generated. A replication run always uses a new stream, and is executed as follows.

1. During the process when the arc costs and solution nodes of the tree are generated, $h^*(n)$ is computed and remembered for each node n.

2. A* is applied to find the cheapest solution path at depth N, using the knowledge of $h^*(n)$. [Of course, the cheapest solution path is found easily since $h(n) = h^*(n)$.]

3. After having determined the cheapest solution path at depth N, the fuzzy distribution of $h(n)$ is introduced by randomly generating $Y(n)$ and subsequently computing $h(n)$ based on $h^*(n)$ for each node n in the tree.

4. A* is applied, and uses only the information of $h(n)$. The number of nodes expanded by A* before the solution node located at depth N is found is then recorded as the result for a replication run and later is used as a data point for computing $f_{cyc}(j|N)$. We modify ϵ only on a run-by-run basis to analyze the effect of $h^*(n) - h(n)$ on $f_{cyc}(j)$. ◄

Figure 3 shows an instance of $f_{cyc}(j|N)$ resulting from our simulation study as a function of ϵ for $b=3$, $N_{max}=N=6$, $(\alpha, \beta) = (50,100)$, and all leaf nodes are solution nodes. In general, this simulation methodology allows $f_{cyc}(j|N)$ to be computed numerically for any N, which, when combined with the knowledge of $P(N)$, allows the numerical computation of $f_{cyc}(j)$, which subsequently can be used in (3) for computing system reliability.

4.3 Computation of System Reliability

As a specific example of the utility of our expert system model, consider a system like the REACTOR monitoring expert system [25, 26] embedded in a nuclear power plant for recommending an appropriate strategy for core-cooling safety during an emergency. The function of the system is to select the most efficient path out of all available paths which can be used to provide core cooling under a real-time constraint. Similar to REACTOR expert system, all available paths form a tree-like structure with all leaf nodes at depth N_{max} being solution nodes. Furthermore, the system uses A* as the conflict-resolution algorithm for selecting a rule to fire in each Match-Select-Act cycle, compute $h(n)$, and adjust the arc costs of the tree dynamically in response to external events. This example system is characterized by the parameters in table 2.

TABLE 2
Parameters of An Example Expert System

b_p	4
b_n	1
$N_{\max}$	6
$P(N_{\max})$	1 (all leaf nodes are solution nodes)
(α, β)	(50,100)
T	1 msec
T_h	10 msec
p_{addr}	0.8
p_{remover}	0.2
m_{rtf}	uniformly distributed over (2,4)
m_{ftr}	uniformly distributed over (1,3)
μ_{tmatch}	1/100 msec
T_a	10 msec
λ_h	1/month = $1/2.5 \cdot 10^9$ msec

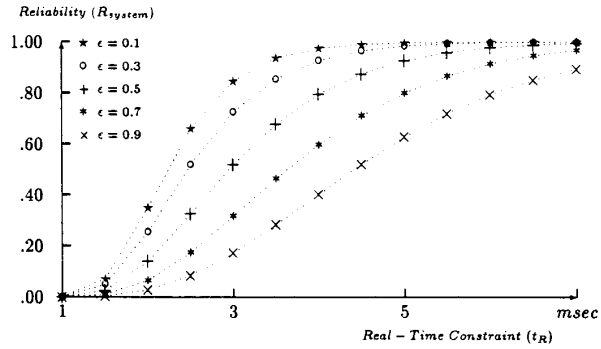

Figure 4. Effect of t_R & ϵ on System Reliability

Figure 4 shows the system reliability as a function of the real-time constraint t_R and the error bound of the heuristic estimates ϵ, using (3) - (5). The expert system can be considered real-time only when the real-time constraint is moderately large (*eg*, $t_R > 4$ msec) and the error bound of heuristic estimates is moderately small (*eg*, $\epsilon < 0.5$).

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under grant CCR-9110816.

REFERENCES

[1] A. Bagchi and A.K. Sen, "Average-case analysis of heuristic search", *Search in Artificial Intelligence* (L. Kanal, V. Kumar, Eds), 1988, pp 131-165; Springer-Verlag.

[2] R.E. Barlow, F. Proschan, *Statistical Theory of Reliability and Life Testing*, 1975; Holt, Rinehart, Winston.

[3] F.B. Bastani, I.R. Chen, "The reliability of embedded AI systems", *IEEE Expert*, vol 8, 1993 Apr, pp. 72-78.

[4] F.B. Bastani, I.R. Chen, T. Tsao, "Reliability of systems with a fuzzy correctness criterion", *Proc. Ann. Reliability & Maintainability Symp*, 1994, pp 442-448.

[5] G. Ciardo, J. Muppala, K. Trivedi, "SPNP: Stochastic Petri net package", *Proc. 3rd Int'l Workshop Petri Nets and Performance Models*, 1989, pp 142-151; CS Press.

[6] A. Cruise, *et al*, "Yes/L1: Integrated rule-based procedural and real-time programming for industrial applications", *Proc. 3rd Conf. Artificial Intelligence Applications*, 1987, pp 134-139.

[7] I.R. Chen, F.B. Bastani, "Effect of artificial intelligence planning-procedures on system reliability", *IEEE Trans. Reliability*, vol 40, 1991 Aug, pp 364-369.

[8] I.R. Chen, B. Poole, "Performance of rule grouping on a real-time expert system architecture", *IEEE Trans. Knowledge & Data Eng*, vol 6, 1994 Dec.

[9] J.B. Dugan, *et al*, "Extended stochastic Petri nets: Applications and analysis", *Performance 84*, (E. Gelenbe, Ed), 1984, pp 507-519; Elsevier.

[10] C.L. Forgy, *OPS5 User's Manual*, CS-81-135, 1981; Carnegie-Mellon Univ.

[11] C.L. Forgy, "Rete: A fast algorithm for the many-pattern/many-object pattern match problem", *Artificial Intelligence*, 1982, pp 17-37.

[12] J.C. Giarratano, *CLIPS User's Guide*, 1988; US L.B. Johnson Space Center.

[13] P.E. Green, "AF: A framework for real-time distributed cooperative problem solving", *Distributed Artificial Intelligence* (M.N. Huhns, Ed) 1987, pp 153-176; Morgan Kaufman.

[14] P.E. Green, *A Data Driven Mechanism for the Execution of Production Rules in Real-Time Computer Based Systems*, 1991; Real-Time Intelligent Systems Corp.

[15] A. Gupta, *Parallelism in Production Systems*, 1987; Morgan Kaufman.

[16] T. Ishida, "Parallel rule firing in production systems", *IEEE Trans. Knowledge & Data Eng*, vol 3, 1991 Mar, pp 11-17.

[17] M. Karnaugh, *et al*, "A computer operator's expert system", *Proc. 7th Int'l Conf. Computer Communications*, 1985, pp 810-815.

[18] M.A. Marsan, G. Conte, G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Trans. Computing Systems*, vol 2, 1984 May, pp 93-122.

[19] W. Mettrey, "A comparative evaluation of expert system tools", *IEEE Computer*, 1991 Feb, pp 19-31.

[20] D.P Miranker, *Treat: A new and efficient match algorithm for AI production systems*, 1989; Pittman/Morgan-Kaufman.

[21] D.P Miranker, B.J. Lofaso, "The organization and performance of a TREAT-based production system compiler", *IEEE Trans. Knowledge & Data Eng*, vol 3, 1991 Mar, pp 3-10.

[22] J.K. Muppala, S.P. Woolet, K.S. Trivedi, "Real-time systems performance in the presence of failures", *IEEE Computer*, 1991 May, pp 37-47.

[23] J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, and Application*, 1987; McGraw-Hill.

[24] P. Nayak, A. Gupta, P. Rosenbloom, "Comparison of the Rete and Treat production matchers for SOAR", *Proc. 6th Nat'l Conf. Artificial Intelligence*, 1988, pp 693-698.

[25] W.R. Nelson, "REACTOR: An expert system for diagnosis and treatment of nuclear reactor accidents", *Proc. 2nd Int'l Conf. Artificial Intelligence*, 1982, pp 296-301.

[26] W.R. Nelson, *Response Trees and Expert Systems for Nuclear Reactor Operations*, 1984 Feb; EG&G Idaho.

[27] J. Pearl, *Heuristics*, 1984; Addison-Wesley.

[28] M.I. Schor, *et al*, "Advances in Rete pattern matching", *Proc. 4th Nat'l Conf. Artificial Intelligence*, 1986, pp 226-232.

[29] P.H. Winston, *Artificial Intelligence* 2nd ed, 1984; Addison-Wesley.

AUTHORS

Dr. Ing-Ray Chen; Inst. of Information Eng'g; National Cheng Kung Univ; No. 1, University Road; Tainan, TAIWAN - R.O.C.

Ing-Ray Chen (S'86, M'90) is an Associate Professor of Information Engineering at the National Cheng Kung University, Taiwan. His research interests are in reliability & performance modeling & analysis, real-time database/expert systems, and application of AI technology to industrial process-control systems. Dr. Chen received the BS (1978) from the National Taiwan University, and the MS (1985) and PhD (1988) in Computer Science from the University of Houston, University Park. Prior to joining National Cheng Kung University, he was an Associate Professor of Computer & Information Science at the University of Mississippi. He is a member of IEEE & ACM.

Dr. Tawei Tsao; Dept. of Computer Science; Weir Hall 302; Univ. of Mississippi; University, Mississippi 38677 USA.

Tawei Tsao received the BS (1983) in Electrical Engineering from the National Taiwan University, and the MS (1989) & PhD (1994) in Computer Science from the University of Mississippi. His research interests are in AI and software reliability engineering.

Manuscript received 1994 January 20.

IEEE Log Number 94-07553 ◄TR►

MANUSCRIPTS RECEIVED *MANUSCRIPTS RECEIVED* *MANUSCRIPTS RECEIVED* *MANUSCRIPTS RECEIVED*

"Importance measures of system components based on uncertainty analysis using fuzzy-set theory" (P. Suresh, V. Raj, et al.), Dr. V. Venkat Raj, Head • Reactor Safety Div. • BARC • Trombay • Bombay - 400 085 • INDIA. (TR94-168)

"A model of a reliability system with s-dependent components" (A. Schottl), Dr. Alfred Schottl • Inst. Applied Mathematics & Statistics SCA • Technical Univ. of Munich • 80290 Munich • Fed. Rep. GERMANY. (TR94-169)

"Point & interval estimation of reliability for 0 failures" (S. Wu), Wu, Shao-min • Dept. of Applied Mathematics • Huaqiao University • Quan Zhou, Fujian - 362 011 • P.R. CHINA. (TR94-171)

"A new reliability assessment method for advanced ceramic tools" (C. Huang, et al.), Chuanzhen Huang • Dept. of Mechanical Eng. • Shandong Univ. of Technology • Jinan, Shandong - 250 114 • P.R. CHINA. (TR94-172)

"Reliability evaluation of process models" (A. Kusiak, et al.), Andrew Kusiak • Intelligent Systems Lab. • The Univ. of Iowa • Iowa City, Iowa 52242-2517 • USA. (TR94-173)

"Distribution of order statistics with varing number of components in a system" (N. Singh, V. Yadavalli), Dr. V. S. S. Yadavalli • Dept. of Statistics & Ops Res • Univ. of the North • PrBag X1106 • Sovenga 0727 • Rep. of SOUTH AFRICA. (TR94-174)

"A hardware/software reliability growth model" (L. Rao), Rao, Lan • Automation Dept. • Tsinghua University • POBox 1021 • Beijing - 100 084 • P.R. CHINA. (TR94-175)

"Operational reliability analysis of modern hardware/software systems" (L. Rao, S. Tong, et al.), Rao, Lan • Automation Dept. • Tsinghua University • POBox 1021 • Beijing - 100 084 • P.R. CHINA. (TR94-176)

"Engineering notions of mean-residual-life functions and hazard-rate functions for finite populations" (N. Ebrahimi), Dr. Nader B. Ebrahimi • Division of Statistics • Dept. of Mathematical Sciences • Northern Illinois University • DeKalb, Illinois 60115-2888 • USA. (TR94-177)

"Downtime of the service station in M/G/1 queueing system with repairable service stations" (Y. Tang), Yinghui Tang • Dept. of Applied Mathematics • Univ. of Electronic Science & Technology • Chengdu, Sichuan - 610 054 • P.R. CHINA. (TR94-178)

"Automatic construction of the Markov transition matrix" (B. Tombuyses), Dr. Ir. Beatrice Tombuyses • Service de Metrologie Nucleaire, CP 165 • Univ. Libre de Bruxelles • 50 Ave F. D. Roosevelt • B-1050 Bruxelles • BELGIUM. (TR94-179)

"Optimal inspection strategies for randomly failing systems" (D. Ait-Kadi, et al.), Dr. Daoud Ait-Kadi • Dept. of Mechanics • Laval Univ. • Quebec G1K 7P4 • CANADA. (TR94-180)

"Optimal number of repairs before replacement for a cold-standby system" (Y. Zhang), Dr. Yuan Lin Zhang • Dept. of Mathematics & Mechanics • Southeast University • Nanjing - 210 018 • P.R. CHINA. (TR94-181)

"Reliability-based optimal task-allocation in distributed database management systems" (A. Verma, et al.), Dr. Ajit K. Verma • Dept. Electrical Engineering (Reliability) • Indian Institute of Technology • Powai, Bombay - 400 076 • INDIA. (TR94-182)

"Capacity constraints in reliability analysis of networks" (A. Abdelaziz), Dr. Ahmed R. Abdelaziz • Dept. of Electrical Engineering • Alexandria University • Alexandria 21544 • EGYPT. (TR94-183)

"The entropy method for reliability estimation of systems of exponential or binomial model components" (J. Shi), Jun Shi • Inst. of Pilotless Aircraft • Nanjing Univ. of Aeronautics & Astronautics • 29 Yudaojei St • Nanjing - 210 016 • P.R. CHINA. (TR94-184)

"Bounds for reliability of 2-dimensional consecutive-k-out-of-r-from-n:F systems" (Z. Psillakis, F. Makri), Dr. Zaharias M. Psillakis • Department of Physics • University of Patras • GR-26110 Patras • GREECE. (TR94-185)

"Statistical analyses of step-stress accelerated life tests" (G. Ge, H. Ma, C. Li), Chunlin Li • Mathematics Section • Hebei Economy Trade College • Shijiazhuang - 050 061 • P.R. CHINA. (TR95-000)

"A Bayes nonparametric framework for software-reliability analysis" (M. El Aroui, J. Soler), Mhamed-Ali El Aroui • IMAG - LMC • BP 53X • 38041 Grenoble cedex 09 • FRANCE. (TR95-001)

"Generalized reliability: Optimal & asymptotic properties with examples" (O. Doctorow), Dr. Osher Doctorow • 2118 Wilshire Blvd, #805 • Santa Monica, California 90403 • USA. (TR95-002)

"Saturated networks" (S. Jain, K. Gopal), S. P. Jain, Assistant Professor • Dept. of Electrical Eng'g • Regional Engineering College • Kurukshetra - 132 119 • INDIA. (TR95-003)