On the Reliability of AI Planning Software in Real-Time Applications

Ing-Ray Chen, *Member, IEEE*, Farokh B. Bastani, *Member, IEEE*, and Ta-Wei Tsao

Abstract— **We define the reliability of a real-time system incorporating AI planning programs as the probability that, for each problem-solving request issued from the environment, the embedded system can successfully plan and execute a response within a specified real-time deadline. A methodology is developed for evaluating the reliability of such systems taking into consideration the fact that, other than program bugs, the intrinsic characteristics of AI planning programs may also cause the embedded system to fail even after all software bugs are removed from the program. The utility of the methodology is demonstrated by applying it to the reliability evaluation of two AI planning algorithms embedded in a real-time multicriteria route-finding system.**

Index Items— **Artificial intelligence (AI), heuristics, AI planning software, real-time, reliability analysis, overall hardware/software system reliability.**

I. Introduction

An AI planning program embedded in a real-time environment [7], [15] typically must respond to an environment request by formulating and executing a plan within a real-time deadline or the system may fail. An example is a combat aircraft system [8] in which an AI planning algorithm is used to search for a path to a target location with the goal of optimizing some specified criteria such as minimizing the probability of radar detection, the distance flown, fuel consumption, etc., subject to the constraint that the total time spent in planning and flying the route must be completed within a real-time deadline.

A design trade-off that exists in such a real-time AI environment is the time invested in planning versus the optimality of the plan formulated by the embedded AI program. As more time is spent in planning, the plan formulated is more likely to be optimal and thus can be executed by the underlying hardware more reliably, e.g., a combat aircraft is less likely to be detected and destroyed when it follows a more optimal route. On the other hand, if too much time is devoted to planning, the mission may also fail because there may not be enough time left for executing the plan within the real-time deadline. This trade-off exists for any real-time system incorporating AI techniques for analyzing combinatorial problems as the time for executing an AI algorithm typically grows exponentially as a function of problem size.

Two approaches have been investigated in the literature to address this trade-off, hoping to make AI techniques more suitable for real-time applications. One approach is to explore parallel or distributed architectures with the goal of reducing the time needed for planning and execution, particularly in the area of production systems [4], [9], [12], [16]. Another approach is to devise real-time or time-constrained AI planning algorithms [10], [14], [23] that attempt to plan (and possibly execute) a response within a specified real-time deadline so that the solution found is near-optimal *most of the time*. For the second approach, there is little work that addresses the associated reliability issue when these AI planning algorithms are applied to real-time systems. It is not satisfactory to rely on informal methods of selecting appropriate designs, given the fact that these AI planning algorithms may work most of the time but sometimes fail miserably [14], [19]. The main issue is a reliability modeling method with which the trade-off between solution optimality and search efficiency in the presence of a real-time deadline can be quantified, and with which the effect of the variance in the planning and execution times on system reliability, as a result of adopting AI planning algorithms, can be accounted for and analyzed. Previous performance analyses of AI planning algorithms, such as A^* [19], anytime algorithms [2], [6], RTA^* [14], TCA^* [23] and *DYNORA* [10], are only for the average time/space behavior and thus are not applicable to reliability modeling.

In our previous work, we investigated a reliability model [3] in which the reliability of a real-time system incorporating an AI planning program *in a single mission* is defined as the probability that the system can successfully accomplish the mission, without causing a software or hardware failure. That is,

$$R_{system} = \int_0^{t_R} \int_0^{t_R - t_p} R_{hardware}(t_p, t_e)\, R_{software}(t_p, t_e)\, dF(t_p, t_e)$$

where t_p is the planning time used by the embedded AI planning program to plan a strategy, t_e is the execution time used by the underlying hardware to carry out the formulated plan, and t_R is the specified real-time constraint for the mission. In the above formulation, the system reliability of the embedded

Ing-Ray Chen is with the Institute of Information Engineering, National Cheng Kung University, Tainan, Taiwan. Ta-Wei Tsao is with the Department of Computer and Information Science, University of Mississippi, Weir 302, University, MS 38677. The work of Ing-Ray Chen and Ta-Wei Tsao was supported in part by the National Science Foundation under Grant CCR-9110816. Farokh B. Bastani is with the Department of Computer Science, University of Houston, Houston, TX 77204-3475. The work of Farokh Bastani was supported in part by the US Nuclear Regulatory Commission under award NRC-04-92-090. The opinions, findings, conclusions, and recommendations expressed herein are the authors and do not necessarily reflect the views of the NRC.

IEEE Log Number K95001.

system, R_{system}, is the product of the hardware reliability, $R_{hardware}$, and software reliability, $R_{software}$, accounting for all possible time distributions of the planning and execution phases.

Hardware failures in a mission are mostly due to stress and wear of the underlying hardware components for carrying out the plans formulated by the software. On the other hand, software failures of AI planning software in real-time applications are caused by (a) residual faults, i.e., bugs in the program, and (b) intrinsic faults of AI software, i.e., the use of heuristic algorithms which have fundamental limitations that may occasionally result in software failures and/or real-time deadline-violation. For example, a bug-free AI program for controlling an automated factory can sometimes assemble products not of the best quality or not in the most optimal way, which in effect can be considered a software failure. Real-time deadline violation failures manifest themselves particularly in AI software because, unlike conventional software, AI software typically must find a solution in a combinatorial search space which involves a tradeoff between the planning time and the execution time. Note that in the above equation, the sum of the variables t_p and t_e must be less than t_R in order to satisfy the real-time requirement – this is indicated by the upper bounds on the double integrals. With this formulation, it is possible to analyze the effect of some AI planning procedures, such as hill climbing and A^* [24], on system reliability [3] based on some assumptions regarding the distributions of t_p and t_e, the software and hardware failure rates, and the reliability of the planning algorithm itself.

This paper extends our previous work with the following specific contributions. First, it proposes a new method for evaluating the reliability of an AI system on a mission by mission basis, rather than on a time basis as has been done in the area of software reliability growth modeling [20], thus unifying this work with our previous work for which only a single mission is considered. The reliability measure on a mission by mission basis is more appropriate for embedded AI systems designed to deal with real-time events each corresponding to a mission for which the system must formulate and execute a control strategy within a real-time deadline specified by the event. An example is an aircraft system incorporating an AI planning program for performing combat missions. This reliability measure can estimate how many such missions the system can accomplish before it fails and is more informative than a time based reliability measure. Specifically, we consider the overall system reliability as a function, $R_{system}(\mathcal{N})$, of the number of missions (or problem-solving requests), $\mathcal{N}$, which the system may encounter during its life time. A method for estimating $R_{system}(\mathcal{N})$ will be presented in the paper. Second, the paper quantifies the failure of AI programs due to intrinsic faults by introducing the concept of fuzzy failure levels. That is, it considers the output of an AI planning program as a fuzzy quantity [25] ranging from 0 to 1 with 0 meaning no failure and 1 meaning a definite failure, rather than treating the output as a binary 0 or 1 quantity as in conventional software. It then considers the possibility that a single heuristic failure may not cause the system to fail but a sequence of heuristic failures may do so. This concept is similar to shock analysis in hardware reliability theory [1]. An example is a combat aircraft system where a plan formulated with radar detection probability of 0.1 may not be catastrophic to the aircraft for that mission, but a number of such missions may cause the aircraft to fail. Third, the paper develops a reliability model that facilitates reliability assessment without having to make any speculative assumption regarding the behavior of the AI system. The idea is based on software reliability modeling where the AI system is tested with its mission profile, through which failure data due to software or hardware failures are collected for estimating the overall system reliability. Lastly, it applies the reliability model to a class of planning algorithms in a practical route-finding system and analyzes the effect of possible design alternatives on the reliability of the system embedding these planning algorithms.

II. Methodology and Model

We first give an abstract description of real-time systems incorporating AI planning programs (for brevity, we will henceforth refer to these as real-time AI systems). We will use the term "mission" interchangeably with "problem-solving request" or "sensor event" in contexts where appropriate. In our model, problem-solving requests arrive at the AI system periodically, each representing a mission for which the system must plan and execute a response within a specified real-time deadline associated with the mission. These problem-solving requests are not known a priori but can be simulated during testing based on the system's anticipated mission profile. An example is a combat aircraft system for which the target and radar locations for a mission are not known but we can simulate missions to test its responses based on its mission profile during the operational period. Our model of a real-time AI system processes these problem-solving requests one at a time.

The point of interest is a method for estimating the system reliability for a specified number of missions. Now,

$$\begin{aligned} &R_{system}(\mathcal{N}) \\ &= P\{\text{no failure over } \mathcal{N} \text{ missions}\} \\ &= P\{\text{no software failure and no hardware failure over } \mathcal{N} \text{ missions}\}. \end{aligned}$$

We assume that the software is run to completion for each mission. Hence, we can adopt the input domain view of software reliability [20]. That is, we can view the software as either failing or not failing for any given point in its input space. This makes the failure probability of the software a function of the number of runs or missions rather than time. Hence, the reliability expression,

$$R_{system} = \int_0^{t_R} \int_0^{t_R - t_p} R_{hardware}(t_p, t_e)\, R_{software}(t_p, t_e)\, dF(t_p, t_e)$$

can be written as

$$R_{system}(1) = R_{software}(1) \int_0^{tR} \int_0^{tR-tp} R_{hardware}(t_p, t_e)\, dF(t_p, t_e)$$

which can be rewritten as

$$R_{system}(1) = R_{software}(1)\, R_{hardware}(1).$$

Assuming that the hardware reliability is 1 in between missions, we get

$$R_{system}(\mathcal{N}) = R_{software}(\mathcal{N})\, R_{hardware}(\mathcal{N}).$$

In the following sections, we develop an approach for estimating $R_{software}(\mathcal{N})$ and $R_{hardware}(\mathcal{N})$.

A. Software Reliability Assessment

To estimate $R_{software}(\mathcal{N})$ as a function of the number of problem-solving requests $\mathcal{N}$, we make the following general observations. First, not all software faults in AI planning algorithms can be corrected since they may be due to limitations in the fundamental techniques used in the algorithms rather than due to coding errors. For example, the *RTA** planning algorithm [14] may find suboptimal solutions and may sometimes even find a solution with a poor solution quality. These software faults are not correctable even after all program bugs are completely removed. Second, some AI planning algorithms guarantee finding optimal solutions, such as the A^* algorithm with underestimated heuristics [24]. However, optimal algorithms are subject to deadline violation failures. A deadline violation failure in this case is not caused by program bugs but rather by the fundamental technique used in the implementation. Third, to model the fact that AI planning algorithms may sometimes output a plan that is not acceptable, for each formulated plan we can associate a failure level which is a fuzzy member [25] of the distribution over the interval [0,1], with 1 representing a definitely unacceptable output and 0 representing a definitely acceptable output. It is often possible to tolerate a few ill-formulated plans occasionally, but not too frequently because of the higher probability of system failure. In general, the assignment of fuzzy levels to plans is to be done during the testing period by experts in the application domain.

A. 1. A Mission-Based Model to Consider AI Intrinsic Faults

In our model, we consider not only program bugs, but also intrinsic faults which exist in AI programs that cause heuristic and deadline-violation failures. We split the failure probability into three independent portions. The first one has a decreasing value during the development phase reflecting the growth in the reliability as a result of program bugs which are detected and removed. The second one has a constant value and is due to heuristic faults which are not removed. The third one also has a constant value and is due to intrinsic faults which cause real-time deadlines to be violated. Thus,

$$\begin{aligned} &R_{software}(\mathcal{N}) \\ &= P\{\text{no coding failure over } \mathcal{N} \text{ runs}, \\ &\text{no heuristic failure over } \mathcal{N} \text{ runs}, \\ &\text{no deadline violation over } \mathcal{N} \text{ runs}\}. \end{aligned}$$

Since these are independent failure processes,

$$\begin{aligned} &R_{software}(\mathcal{N}) \\ &= P\{\text{no coding failure over } \mathcal{N} \text{ missions}\} \times \\ &P\{\text{no heuristic failure over } \mathcal{N} \text{ missions}\} \times \\ &P\{\text{no deadline violation over } \mathcal{N} \text{ missions}\} \\ &= (1-\lambda_1)^{\mathcal{N}} \times (1-\lambda_2)^{\mathcal{N}} \times (1-\lambda_3)^{\mathcal{N}} \end{aligned}$$

where $\lambda_1 = P\{\text{coding failure over 1 run}\}$, $\lambda_2 = P\{\text{heuristic failure over 1 run}\}$, and $\lambda_3 = P\{\text{deadline violation over 1 run}\}$. For large $\mathcal{N}$ and small λ_1, λ_2, and λ_3, the geometric distribution in the above expression can be approximated by the corresponding exponential distribution:

$$R_{software}(\mathcal{N}) = e^{-\lambda_1 \mathcal{N}} \times e^{-\lambda_2 \mathcal{N}} \times e^{-\lambda_3 \mathcal{N}}$$

During the software testing and debugging phase, the implemented program is tested with its mission profile until a failure is encountered. If the failure is due to program bugs, the fault is located and removed from the program. The *failure history* of the AI program is defined to be the realization of a sequence of random variables $(N_1, N_2, \ldots, N_n)$, where N_i denotes the mission number for which a failure is detected. For example, if the system is tested with 10 test cases, each representing a mission, and test cases #2 and #5 each cause a particular type of failure to be detected, then the failure history for that particular failure type will be (2, 5). Hence, the unit in our model is mission, rather than time as in most software reliability growth models [20].

We use a modified version of the Musa-Okumoto Logarithmic model [17] to estimate λ_1. The probability of failure due to program bugs after the i^{th} mission is assumed to be given by

$$\lambda_1(i) = \frac{\lambda_0}{\lambda_0 \theta_i + 1} \tag{1}$$

where λ_0 and θ are model parameters, representing, respectively, the initial failure probability of the program and the relative change of failure probability per failure experienced and removed. Equation (1) models the fact that the program failure probability due to residual program bugs decreases continuously over the testing and debugging phase, rather than at discrete event-points corresponding to failure detection and removal times. Further, the rate of decrease in $\lambda_1(i)$ itself decreases as more test cases are tested, thus modeling the decrease in the size of errors detected as debugging proceeds.

We model the unchanging failure probabilities due to intrinsic faults as follows:

$$\lambda_2(i) = \lambda_2 \text{ and } \lambda_3(i) = \lambda_3,$$

where $\lambda_2(i)$ and $\lambda_3(i)$ are constant failure probabilities due to intrinsic faults associated with AI planning programs. $\lambda_2(i)$ is due to heuristic faults while $\lambda_3(i)$ is due to deadline-violation faults. We distinguish between heuristic and deadline-violation

faults because the latter, unlike the former, involve a trade-off between time and accuracy. Heuristic faults are generally due to the use of heuristic search procedures and can result in suboptimal decisions even though the search procedure is executed fully. On the other hand, to deal with real-time constraints, a suboptimal decision may have to be accepted simply because there is no time to fully execute the search procedure. If the program is run on a faster machine, failures due to real-time violations will decrease while those due to heuristic faults will remain unchanged.

During the operational phase, no program bug is removed, so that $\lambda_1(i)$ is also constant, say λ_1. Thus, the reliability during this phase is given by

$$R_{software}(\mathcal{N}) = e^{-(\lambda_1+\lambda_2+\lambda_3)\mathcal{N}} \tag{2}$$

The above equation for software reliability estimation is pessimistic because it considers a heuristic failure as a definite failure so that when a heuristic failure occurs the system also fails. There are systems that can accept suboptimal solutions (which are fuzzy in the sense that the solution found may not be the best possible solution) as long as the mission in a planning and execution cycle can be accomplished within the real-time deadline. This fuzzy situation can be modeled by considering the distribution of heuristic failures over [0,1], with 0 denoting a benign (i.e., no) failure and 1 denoting a definite failure. Heuristic faults due to the underlying approach are assumed to have a distribution G, with $G(0) = 0$ and $G(1) = 1$. The *level of service* provided by the program can be measured in several ways, one of which is discussed in the following.

We consider the system as having failed if the sum of all heuristic failures encountered exceeds some limit, say X_L. This models situations where the effect of failures can accumulate. An example is in the problem of route planning where the route is decomposed into several segments. In this case, if there is a slight error in a segment, then it can be tolerated. However, the errors in the segments can accumulate resulting in a substantial and unacceptable error in the overall path. Another example is for a combat aircraft system designed to deal with a sequence of missions. The aircraft system may tolerate some plans each with a radar detection probability not equal to 0, but if the sum of such radar detection probabilities is greater than 1, the system is likely to fail eventually.

Let X_i be a random variable indicating the level of the i^{th} heuristic failure. Then, during the system's operational phase, the software reliability is given by

$$\begin{aligned}
&R_{software}(\mathcal{N}) \\
&= P\{\text{software is alive after } \mathcal{N} \text{ missions}\} \\
&= P\{\text{no failure due to bugs or deadline-violation is experienced}\} \times \\
&\qquad P\{\text{accumulated heuristic failures experienced} \le X_L\} \\
&= e^{-(\lambda_1+\lambda_3)\mathcal{N}} \times \sum_{n=0}^{\infty} Pr\{n \text{ heuristic failures over } \mathcal{N} \text{ missions}\}\, Pr\{X_1 \\
&\qquad +\ldots+X_n \le X_L\} \\
&= e^{-(\lambda_1+\lambda_3)\mathcal{N}} \times \sum_{n=0}^{\infty} \frac{e^{-\lambda_2\mathcal{N}}(\lambda_2\mathcal{N})^n}{n!} G^{(n)}(X_L)
\end{aligned} \tag{3}$$

where n is the total number of heuristic failures which can probabilistically occur in $\mathcal{N}$ missions, and $G^{(n)}(x)$ denotes the n-fold convolution of $G(x)$, representing the probability that the sum of n random variables of $G(\cdot)$ is less than x. It is defined as

$$G^{(n)}(x) = \begin{cases} 1 & \text{if } n=0 \\ G(x) & \text{if } n=1 \\ \int_0^x G^{(n-1)}(x-y)\,dG(y) & \text{if } n \ge 1 \end{cases}$$

where $G(x)$ is the probability of $X \le x$ for a random variable X that represents the fuzzy level of a heuristic failure.

A. 2. Estimation of Model Parameters

Four sets of failure data are required in order to estimate the parameters of Equation (3). These are $(N_{c1}, N_{c2},\ldots, N_{cs})$ (for failures caused by program bugs), $(N_{h1}, N_{h2},\ldots, N_{hr})$ (for heuristic failures), $(f_1, f_2,\ldots, f_r)$ (for associated fuzzy levels), and $(N_{d1}, N_{d2},\ldots, N_{dq})$ (for failures cause by deadline-violation), where N_{ci} is the mission number for which the i^{th} failure caused by program bugs is found and the corresponding program bugs are removed; N_{hi} is the mission number for which the i^{th} heuristic failure is found; N_{di} is the mission number for which the i^{th} deadline-violation failure is found; and f_i is the fuzzy level of the i^{th} heuristic failure. These failure data may be obtained during the testing and debugging phase through testing the AI system with its anticipated mission profile.

The maximum likelihood estimates (MLEs) can be derived separately for each of the three independent failure processes. Let us first consider the MLEs for λ_0 and θ which are parameters of the probability of failure due to coding faults. The probability density function $pdf(i)$ for the event "there is no failure between runs $N_{c(i-1)}$ and N_{ci} and there is a failure on run N_{ci}" is given by:

$$\begin{aligned}
pdf(i) &= \left[\prod_{j=N_{c(i-1)}}^{N_{ci}-2}\left(1-\lambda_1(j)\right)\right]\lambda_1(N_{ci}-1) \\
&\approx e^{-\sum_{j=N_{c(i-1)}}^{N_{ci}-2}\lambda_1(j)}\lambda_1(N_{ci}-1) \quad \text{for small } \lambda_1(j).
\end{aligned}$$

Now,

$$\sum_{j=N_{c(i-1)}}^{N_{ci}-2}\lambda_1(j) = \sum_{j=1}^{N_{ci}-2}\lambda_1(j) - \sum_{j=1}^{N_{c(i-1)}-1}\lambda_1(j)$$

$$\sum_{j=1}^{N_{ci}-2}\lambda_1(j) = \sum_{j=1}^{N_{ci}-2}\frac{\lambda_0}{\lambda_0\theta_j+1} \approx \frac{1}{\theta}log(\lambda_0\theta N_{ci}+1).$$

Hence,

$$pdf(i) \approx e^{-\frac{1}{\theta}\log\frac{\lambda_0\theta N_{ci}+1}{\lambda_0\theta N_{c(i-1)}+1}} \frac{\lambda_0}{\lambda_0\theta N_{ci}+1} \approx \lambda_0 \frac{\left(\lambda_0\theta N_{c(i-1)}+1\right)^{\frac{1}{\theta}}}{\left(\lambda_0\theta N_{ci}+1\right)^{\frac{1}{\theta}+1}}$$

From this, approximate values of the maximum likelihood estimates of λ_0 and θ can be obtained by numerically solving the following equations [17]:

$$\frac{s}{\hat{\lambda}_0} - \hat{\theta}\sum_{j=1}^{s}\frac{N_{cj}}{\hat{\lambda}\hat{\theta}N_{cj}+1} - \frac{N_{cs}}{\hat{\lambda}_0\hat{\theta}N_{cs}+1} = 0;$$

$$-\hat{\lambda}_0\sum_{j=1}^{s}\frac{N_{cj}}{\hat{\lambda}_0\hat{\theta}N_{cj}+1} + \frac{1}{\hat{\theta}^2}\log\left(\hat{\lambda}_0\hat{\theta}N_{cs}+1\right) - \frac{\hat{\lambda}_0 N_{cs}}{\hat{\theta}\left(\hat{\lambda}_0\hat{\theta}N_{cs}+1\right)} = 0 \tag{4}$$

The failure probability density of heuristic failures on a mission by mission basis is

$$PDF_{fi}(\mathcal{N}) = \lambda_2 e^{-\lambda_2 \mathcal{N}}$$

Therefore, the maximum likelihood estimate [17] of λ_2 due to heuristic faults can be estimated as

$$\hat{\lambda}_2 = \frac{r}{N_{hr}} \tag{5}$$

where r is the total number of heuristic failures experienced during the testing period, and N_{hr}, defined as before, is the mission number for which the r^{th} heuristic fault is experienced. For example, if (#2, #8, #25) are a set of 3 missions for which heuristic failures are detected during the testing phase, then λ_2 will be calculated as 3/25. In other words, the system can experience a heuristic failure once in about every 8 missions when the system is in its operational phase.

The maximum likelihood estimate of λ_3 due to deadline-violation can be estimated in a similar way as

$$\hat{\lambda}_3 = \frac{q}{N_{dq}} \tag{6}$$

where q is the total number of deadline-violation failures experienced during the testing and debugging period.

A reasonable model for G is the Beta(α, β) distribution[1] [11] with density

$$g(x) = \begin{cases} \dfrac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

[1] A Beta distribution is defined as follows: if X and Y are independent Gamma random variables with parameters (α, λ) and (β, λ), respectively, then the joint density of $X/(X+Y)$ is called the beta density with parameter (α, β).

The maximum likelihood estimates of α and β can be obtained by numerically solving the following equations using the fuzzy level data set $(f_1, f_2, ..., f_r)$ collected during the testing phase:

$$r\frac{\frac{\partial\Gamma(\hat{\alpha}+\hat{\beta})}{\partial\hat{\alpha}}}{\Gamma(\hat{\alpha}+\hat{\beta})} - r\frac{\frac{\partial\Gamma(\hat{\alpha})}{\partial\hat{\alpha}}}{\Gamma(\hat{\alpha})} + \sum_{i=1}^{r}\log f_i = 0$$

$$r\frac{\frac{\partial\Gamma(\hat{\alpha}+\hat{\beta})}{\partial\hat{\beta}}}{\Gamma(\hat{\alpha}+\hat{\beta})} - r\frac{\frac{\partial\Gamma(\hat{\beta})}{\partial\hat{\beta}}}{\Gamma(\hat{\alpha})} + \sum_{i=1}^{r}\log(1-f_i) = 0 \tag{7}$$

where

$$\frac{\partial\Gamma(\hat{\alpha}+\hat{\beta})}{\partial\hat{\alpha}} = \int_0^\infty (\log x) x^{\hat{\alpha}+\hat{\beta}-1} e^{-x} dx.$$

A less general, though simpler model, is to consider a single parameter Beta distribution with α equal to 1. In this case, the density is $\beta(1-x)^{\beta-1}$ for $0 \le x \le 1$ and 0 otherwise. The maximum likelihood estimate of β is

$$\hat{\beta} = \frac{r}{\sum_{i=1}^{r}\log\left(\frac{1}{1-f_i}\right)}.$$

B. Hardware Reliability Assessment

In this section, we develop a method for estimating $R_{hardware}(\mathcal{N})$ as a function of the number of missions $\mathcal{N}$. We again assume that during the testing and debugging phase, the system is tested with the anticipated mission profile so that each test case corresponds to a mission for which a control plan formulated by the embedded AI program is executed by the underlying hardware.

B. 1. Collecting Hardware Failure Data

For any mission, the hardware system can be considered to have failed if the estimated hardware reliability of the mission is less than a specified hardware reliability requirement, e.g., 0.9_i where the notation 0.9_i represents a hardware reliability requirement having i 9's to the right of the decimal point [13]. Such hardware failure data can be collected during the testing phase on a mission by mission basis. That is, for each mission we (a) first compute the hardware reliability of the planning phase and the hardware reliability of the execution phase, respectively; (b) then multiply these two quantities to obtain the hardware reliability of the mission; and (c) then compare the hardware reliability of the mission with the specified hardware reliability requirement to determine if the mission has caused a hardware failure. The hardware failure data set collected this way is $(N_{h1}, N_{h2}, ..., N_{hw})$ where w is the total number of hard-

ware failures experienced during the testing phase, and N_{hj}, $1 \leq j \leq w$, is the mission number for which a hardware failure is experienced.

In the following, we outline a procedure for collecting hardware failure data. Suppose that for the j^{th} test case (mission) the system spent time t_p^j to plan a strategy, then the hardware reliability of the planning phase for the j^{th} mission can be calculated as $e^{-\lambda_p t_p^j}$ where λ_p is the failure rate (in unit of failures/time) of the computer system on which the embedded AI planing program is run, assuming that the failure time of the computer hardware is exponentially distributed with a constant rate of λ_p. In hardware reliability theory, it is common to assume that a hardware component obeys this "exponential failure" law during its useful life period [13]. Of course, other distributions such as the Weibull distribution [22] can also be used to model the computer system if justified.

The hardware reliability of the execution phase of the j^{th} mission, on the other hand, can be calculated based on the plan formulated by the embedded AI planning procedure in response to the j^{th} mission, using standard hardware reliability assessment techniques [13], [22]. For example, if the embedded system is a robot system and the formulated plan is to move its hand, arm and leg hardware actuators simultaneously to reach for an object such that as long as two actuators remain alive mission j can be accomplished, then the hardware reliability of the execution phase is that of a 2-out-of-3 parallel system for the duration of the execution phase for mission j, a quantity which can be computed easily since the component reliabilities of the hand, arm and leg are normally known a priori, e.g., each obeying the exponential failure law with a distinct constant failure rate. On the other hand, if the formulated plan is to move the leg actuator first and subsequently the hand actuator after the leg motion has come to a stop, then the hardware reliability of the execution phase is that of a series system connecting the leg and hand components, which again can be computed easily. For a vehicle system, we can first compute the total execution time needed to execute mission j, say, t_e^j. Then, by assuming that the vehicle underlying hardware also obeys the exponential failure law with a constant failure rate of λ_e, the hardware reliability of the execution phase can be computed as $e^{-\lambda_e t_e^j}$.

B. 2. Computing Hardware Reliability

It is important to note that hardware failures experienced during the testing and debugging period are partly determined by intrinsic faults of AI programs which, through planning, implicitly determine the execution time as well as the structure function of the underlying hardware system. Let λ_h denote the constant hardware failure rate of the embedded system (in unit of failures/mission) due to uncorrectable intrinsic faults of AI planning programs. Then, the maximum likelihood estimate of λ_h may be estimated as

$$\hat{\lambda}_h = \frac{w}{N_{hw}}$$

where w, defined as before in Section B.1., is the total number of hardware failures collected during the testing phase, and N_{hw} is the mission number for which the last hardware failure is observed. The hardware reliability of the embedded system in the operational phase then can be calculated as

$$R_{hardware}(\mathcal{N}) = e^{-\lambda_h \mathcal{N}}. \tag{8}$$

Multiplying $R_{hardware}(\mathcal{N})$ with $R_{software}(\mathcal{N})$ obtained earlier in Equation (3) in Section A.1 yields the overall system reliability $R_{system}(\mathcal{N})$ as a function of the number of missions $\mathcal{N}$.

III. Application

In this section, we show the application of the methodology and model developed in Section II. The objective is to demonstrate how the model can be used to evaluate the reliability of a real-time system incorporating AI planning algorithms.

A. Description of the Case Study

The real-time system used in our case-study is a version of the multicriteria aircraft routing system reported in [8] where an AI planning algorithm embedded in the system is used to plan and follow a path across radar-threating areas to reach a target location within a specified real-time deadline. Missions are assigned one at a time. The locations of the target and radars, the detection intensity of each radar, and the real-time deadline are not known until a mission is assigned.

For analysis purposes, a two-dimensional version of the system, i.e., a multicriteria *vehicle* routing system, is considered. Each mission is characterized by (a) 50-100 routing points arbitrarily located on a Cartesian plan with arbitrary connections between them, and (b) 3 radars randomly located between the (randomly generated) start and target locations, and (c) a real-time deadline of 2 minutes to reach the target; the targets are mobile and can move beyond the target zone in 2 minutes. The cost of the distance between two connected routing points is in the interval of [10, 100] in unit of feet. Each radar is capable of detecting the vehicle with a circular detection intensity inversely proportional to the distance between the vehicle and the location of the radar. The radar detection intensity has the unit of radar detection probability per foot to account for the fact that traveling a longer route with a nonzero radar detection probability intensity will incur a higher detection probability than traveling a shorter route with the same radar detection intensity. The radar detection intensity of a connecting edge with respect to a radar is in the range of [0, 0.005], depending on how far the edge is away from the location of the radar.

Note that our reliability model allows each mission to be associated with its own deadline. Our deadline assumption of 2 minutes is arbitrary and does not affect the way the fuzzy failure level of a mission is calculated. For the vehicle routing system, the fuzzy failure level of a mission is the cumulative probability of radar detection during that mission's execution phase and, therefore, includes the effect of the execution time

(e.g., 1 minute vs. 1 hour) in the mission. It is thus a measure indicating the probability of a mission failure, regardless of whether the mission is short or long. We also note that when the vehicle is idle (i.e., in between missions), the system can be considered reliable with probability 1, i.e., during the time in which the system is idle or between missions, the reliability of the system is 1.

Fig. 1 shows an example routing map with radar threats. The mission profile generated this way is assumed to correspond to the operational profile when the system is released for operational use.

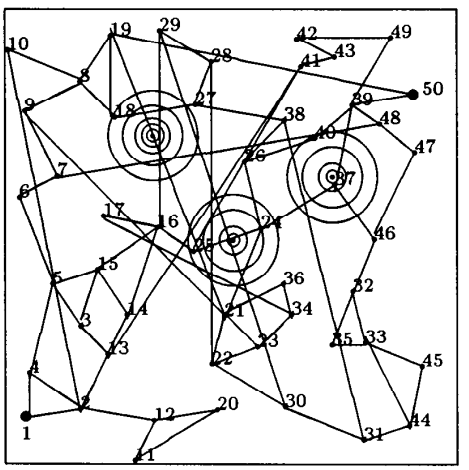


Fig. 1. A Routing Map with Radar Threats (1: Start; 50: Target).

We next give a brief description of the multicriteria vehicle routing system itself. The objective of the system is to plan and execute a route for each mission with a cumulative cost as small as possible without violating the specified real-time deadline $t_R = 2$ minutes. Two criteria for the cost function are the radar detection probability and the total distance traveled, the relative weight of which is determined by a design parameter called w_{radar}. Adopting the cost function definition in [8], the cost between any two points with a direct edge is given by

$$cost = w_{radar} \times p_{radar} \times distance + (1 - w_{radar}) \times distance \quad (9)$$

where w_{radar} is the relative weight for the radar detection criterion, p_{radar} is the radar detection probability of the edge, and *distance* is the true distance of the edge in feet. When w_{radar} is equal to 1, the major concern is to minimize the radar detection probability. If the vehicle must execute a plan with a radar detection probability greater than 0, then the plan is considered unsafe, corresponding to the case that a heuristic failure has been experienced with the radar detection probability of the route planned equal to its fuzzy level. Furthermore, in analyzing the effect of different AI planning algorithms on system reliability, it is assumed that if the sum of accumulated fuzzy failure levels exceeds 1, the system is considered to have failed. Each failure poses a certain amount of risk to the operators and only a fixed number of such exposures to danger can be tolerated before considering the system as unacceptable.

The hardware characteristics of the vehicle system are described in the following. The vehicle can travel a route with a speed of 10,000 feet/minute. The failure time of the underlying hardware components of the vehicle due to wear and tear is assumed to be exponentially distributed with a constant failure rate of $\lambda_e = 3 \times 10^{-5}$ failure/minute. (About once in every 20 days of continuous operations.) The computer system embedded in the vehicle to run an AI planning program is assumed to have a failure time exponentially distributed with a constant failure rate of $\lambda_p = 5 \times 10^{-6}$ failures/minute (about once in every 100 days of continuous operations), and can process (expand or generate) about 2,000 nodes per minute. The hardware reliability requirement of any mission is 0.99999. Therefore, suppose a mission requires t_p planning time and the total distance (not the cost) of the route planned is C feet, then the system is considered to have experienced a hardware failure for that mission if

$$e^{-\lambda_p t_p} \times e^{-\lambda_e \frac{C}{10000}} < 0.99999.$$

B. Possible Planning Programs and Design Alternatives

Given the description for the mission profile and the vehicle routing system, a system designer is faced with the problem of evaluating different design options for improving the reliability of the system. Some possible design options are given below.

- Option 1 is to use an *optimal* search algorithm such as A^* with *lower-bound estimates* [24] as the underlying planning algorithm. A^* operates by ranking all partially explored routes by the accumulated cost so far (g) and a lower-bound estimate of the cost remaining (h), and always expanding the route with the minimum $f = g + h$ value among all routes until the target is found. Because the estimate of the remaining cost of a partially explored route can be underestimated, i.e., using the linear cost between the frontier point of a partially explored route and the target location with the radar detection probability set to 0, the final route planned is optimal [21], [24], implying that the route planned should incur a smaller radar detection probability. Nevertheless, a potential problem with A^* is that the system may not be able to plan and execute a solution within the real-time deadline t_R due to excessive planning. In [8], A^* was selected as the planning algorithm for the aircraft routing system. One can imagine that the system operating under A^* first plans a route, and then the vehicle executes the route in one execution step to reach the target location.
- The second option is to avoid deadline-violation failures as much as possible by using a planning algorithm that allows the target to be reached more quickly, although perhaps not through the best possible route with the minimum cost function defined in Equation (9). An approach is to impose a maximum planning time interval within which an embedded AI planning algorithm must

formulate a solution. However, this approach is practical only for applications where a low quality solution is available initially which can be improved over time by using a class of planning algorithms called *anytime* algorithms [6]. For our vehicle routing system, since the vehicle system must follow existing routes on the ground, there is no immediate solution to begin with. The only practical way is to try to plan and execute a solution within the real-time deadline constraint since it is useless to set a maximum time constraint on the planning phase alone. There are many such real-time planning algorithms reported in the literature over the past five years, such as RTA^* [14], TCA^* [23], and *DYNORA* [10].

- Without loss of generality, this paper considers RTA^*; other real-time algorithms can be analyzed using our model in a similar way. As in A^*, RTA^* also expands the route with the minimum $f = g + h$ value. However, unlike A^*, RTA^* at all time only expands one route until the target location is reached. It nullifies actions by allowing backtracking so that it can revisit a previously visited location if it has to. When it decides the next routing point to move, it actually executes the move to change the current location of the vehicle, thus extending the route it expands. Therefore, the routing point is measured relative to the current location of the vehicle, and the initial starting location of the vehicle is irrelevant. One can imagine that the vehicle system operating under RTA^* will repeatedly plan and move in cycles until the target location is reached. In each cycle, it generates neighboring locations from the current location and expands the one with the minimum $g + h$. The h value of a neighbor is computed such that if the neighbor was visited before, the h value is looked-up from a hash table that updates the h values of all visited nodes; otherwise, a heuristic evaluation function, possibly augmented by lookahead search, is used to compute the h value [14].
- The advantage of RTA^* is that the system may experience fewer real-time deadline-violation failures. However, more heuristic failures may occur because routes planned may have a higher radar detection probability when compared with those planned by A^*. In this paper, we will use our model to analyze how RTA^* can affect the reliability of the vehicle system described above for the case when the heuristic evaluation function is the linear cost to the target location with w_{radar} set to 0 (as is the case for A^*), and is not augmented by lookahead search for computing h. It should be noted that having no lookahead search is a special case when the search horizon is equal to 0 – it still allows the vehicle to revisit previously visited locations to undo actions. Also, our reliability model is applicable to other cases, so it is a potential tool to analyze the effect of search horizon (= 0 or > 0) on system reliability.
- The third option is to use a faster computer system for planning, say, twice as powerful, hoping to reduce the planning time, thus leaving sufficient time for execution. The analysis of this option requires no rerun of the mission profile. The same set of output data under A^* or RTA^* can be used to deduce the new failure data by considering the fact that the time for expanding a node will only require half as much. Also, using a faster computer machine for planning will only change deadline-violation failure data, since a planning algorithm will still plan the same route for the same mission, regardless of whether the computer on which it runs is more powerful or not. Our model can answer this simple "what-if" type of modification question without having to rerun the mission profile.

C. Experiments and Interpretation of Results

Two real-time vehicle systems incorporating A^* and RTA^*, respectively, were simulated, based on the descriptions given in the previous two sections. Each system was tested with 10,000 missions and software and hardware failure data were collected during the testing and debugging phase. These failure data, one set for each system, are too large to be included in this paper and can be found in [5]. We note that even A^* may occasionally incur heuristic failures, although the frequency of this is not as high as RTA^*. This is because for the vehicle system, the fuzzy level of a plan is defined as the total radar detection probability of the route planned. Consequently, even A^* is not able to plan a perfect route without any radar detection probability all the time. (Note: the frequency of heuristic failures is largely determined by the definition of the cost function in Equation (9).) The failure data for each system were subsequently used to estimate model parameters λ_1, λ_2, λ_3, α, β and λ_h, based on the procedure given in Section 2. Finally, the system reliability is computed as a function of the number of missions $\mathcal{N}$.

Fig. 2 shows the overall system reliability of the vehicle systems operating under A^* and RTA^*, respectively, as a function of the number of missions $\mathcal{N}$. Figure 2 shows that, with the failure conditions defined in Section III. A., a vehicle system incorporating RTA^* can provide a better reliability than a system running A^* when the number of missions $\mathcal{N}$ is less than 2600. Before that cross-over point, deadline-violation failures dominate heuristic failures, thereby favoring RTA^* which incurs fewer deadline-violation failures because it plans more quickly. However, after the cross-over point, RTA^* no longer can provide a better reliability than A^* because RTA^* reaches the point when the sum of fuzzy failure levels exceeds 1 more quickly since it incurs heuristic failures more frequently than A^* due to less optimal routes planned. Nevertheless, for practical purposes, the low reliability (i.e., 0.8) after the cross-over point is not acceptable for a real-time system design, implying that RTA^* is still a viable approach, especially for the vehicle system designed to operate for a finite number of missions. An important observation is that neither planning algorithm can provide the vehicle system with the best reliability for all situations (in terms of the number of missions) because there is a trade-off between the deadline-violation and heuristic failures. For a vehicle system designed to operate indefinitely, neither algorithm can provide a good reliability since the system will eventually fail due to heuristic faults.

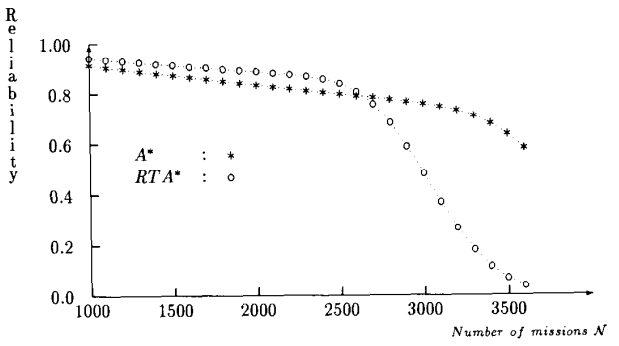


Fig. 2. Reliability Comparison of A^* and RTA^*.

Fig. 3 shows the improvement in reliability when a computer system twice as powerful is used for planning. The reliability of either planning algorithm improves as less time is required for planning when compared with the original computer system. The same cross-over behavior is still observed. However, using a more powerful computer seems to benefit A^* more because a reduced planning time helps reduce deadline-violation failures, but has no effect on heuristic failures which affect RTA^* even more. As a result, the cross-over point is also moved slightly from 2600 to 2500 missions.

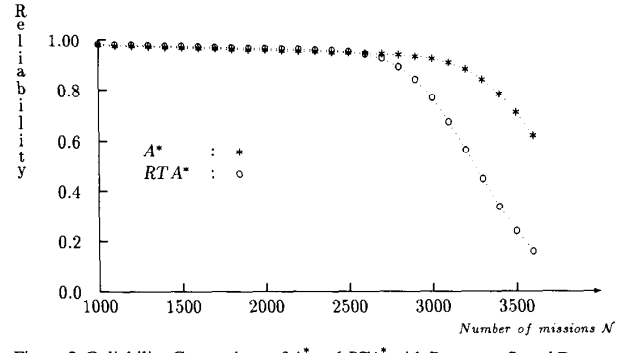


Figure 3. Reliability Comparison of A^* and RTA^* with Processor Speed Doubled.

IV. Summary

In this paper, we first discussed why intrinsic faults of real-time AI planning programs may cause heuristic and deadline-violation failures which are not removable even after the AI program has been tested and debugged for a long period of time. Then, we developed a mission-based method for estimating the reliability of real-time systems incorporating AI planning programs, where we consider failures due to program bugs and deadline-violation are definite mission failures, while failures due to imperfect planning are just heuristic failures, which only cause a definite mission failure if the sum of the accumulated fuzzy failure levels exceeds 1. Reliability evaluations of a real-time vehicle routing system incorporating A^* and RTA^* quantitatively revealed the trade-off between deadline-violation and heuristic failures: A^* can avoid heuristic failures but may incur more deadline-violation failures, while RTA^* is the other way around. Therefore, there exists a cross-over point in the number of missions beyond which one planning algorithm is better than the other.

It is worth mentioning that the exact cross-over point is sensitive not only to the definition of the mission profile, but also to the definition of failure conditions. Here we have defined the sum of accumulated fuzzy levels experienced greater than 1 as a definite failure condition for heuristic failures. One can imagine that a different definition, say, a heuristic failure with a fuzzy level greater than 0.5 is considered a definite failure, may change the location of the cross-over point. Also, a definition based on the product of fuzzy failure levels, rather than the sum, can give a different result. It is therefore important in defining correct failure conditions before applying the model developed in this paper. Our method, however, is universally applicable for evaluating the effect of different design alternatives on system reliability. A possible future extension of this work is to analyze the effect of parallel planning algorithms on the system reliability of production systems, e.g., distributed algorithms with rule partitioning [4], parallel Rete or Treat matching algorithms [9], [16], and parallel rule firing algorithms [12].

References

[1] R.E. Barlow, F. Proschan, *Statistical Theory of Reliability and Life Testing*, New York: Holt, Rinehart and Winston, Inc., 1975.

[2] M. Boddy, "Anytime problem solving using dynamic programming," *9th National Conf. Artificial Intelligence*, pp. 738-743, 1991.

[3] I.R. Chen and F.B. Bastani, "Effect of Artificial Intelligence planning-procedures on system reliability," *IEEE Trans. Reliability*, pp.364-369, Aug., 1991.

[4] I.R. Chen and B. Poole, "Performance of rule grouping on a real-time expert system architecture," *IEEE Trans. Knowledge and Data Eng.*, vol. 6, no. 6, pp. 883-891, Dec., 1994.

[5] I.R. Chen, F.B. Bastani and T.W. Tsao, "On the Intrinsic Faults of Real-Time AI Planning Programs," Technical Report UMCIS-1993-10, Dept. of Computer Science, University of Mississippi, 1993.

[6] T. Dean, and M. Boddy, "An analysis of time-dependent planning," *7th National Conf. Artificial Intelligence*, pp. 49-54, August, 1988.

[7] A.C. Diaz, *An Overview of Realtime Expert Systems*, National Research Council Canada, ERA-380, NRC No. 31759, April 1990.

[8] J. J. Grimm, G.B. Lamont, and A.J. Terzuoli, "A parallelized search strategy for solving a multicriteria aircraft routing problem," *Proc. 1993 ACM/SIGAPP Symposium on Applied Computing*, Indianapolis, pp. 570-577, 1993.

[9] A. Gupta, *Parallelism in Production Systems*, Los Altos: Morgan Kaufman, 1987.

[10] B. Hamidzadeh, and S. Shekhar, "DYNORA: A real-time planning algorithm to meet response-time constraints in dynamic environments," *Proc. 3rd Int. Conf. Tools for AI*, San Jose, pp. 228-235, 1991.

[11] P.G. Hoel, S.C. Port, and C.J. Stone, *Introduction to Probability Theory*, Boston: Houghton Mifflin Co., 1971.

[12] T. Ishida, "Parallel rule firing in production systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 3, no. 1, pp.11-17, March, 1991.

[13] B.W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley, 1989.

[14] R.E. Korf, "Real-time heuristic search," *Artificial Intelligence Journal*, vol. 42, pp. 189-211, 1990.

[15] T.J. Laffey, P.A. Cox, J.L. Schmidt, S.M. Kao and J.Y. Read, "Real-time knowledge based systems," *AI Magazine*, pp. 27-45, Spring 1988.

[16] D.P Miranker, and B.J. Lofaso, "The organization and performance of

a TREAT-based production system compiler," *IEEE Trans. Knowledge and Data Eng.*, vol. 3, no. 1, pp. 3-10, March 1991.

[17] J.D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," *Proc. 7th Int. Conf. Soft. Eng.*, Orlando, FL., pp. 230-237, Mar. 1984.

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

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

[20] C.V. Ramamoorthy, and F.B. Bastani, "Software reliability - status and perspective," *IEEE Trans. Soft. Eng.*, pp. 354-371, July 1982.

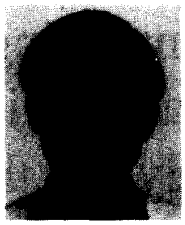
[21] E. Rich, *Artificial Intelligence*, 2nd. Ed., McGraw-Hill, 1991.

[22] S. M. Ross, *Introduction to Probability Models*, 4th Ed., Academic Press, 1989.

[23] B.W. Wah, and L.C. Chu, "*TCA** - A time constrained approximation A^* search algorithm," *Proc. 2nd Inter. Conf. Tools for AI*, Washington D.C., pp. 314-320, 1990.

[24] P.H. Winston, *Artificial Intelligence*, 2nd Edition, Addison-Wesley, 1984.

[25] L.A. Zadeh, "Fuzzy sets and information granularity," *Advances in Fuzzy Set Theory and Application* (edited by M.M. Gupta, R.D. Ragade, and R.R. Yager), North-Holland, 1979.



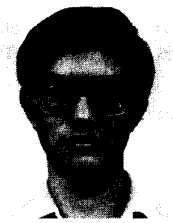
Ing-Ray Chen (M'89) received the B.S. degree from the National Taiwan University, and the M.S. and Ph.D. degrees in computer science from the University of Houston, TX.

He is currently an associate professor of Information Engineering, National Cheng Kung University, Tainan, Taiwan. His research interests are in reliability and performance analysis, and real-time intelligent systems. D. Chen is a member of the IEEE, ACM, and AAAI.

Farokh Bastani (M'82) received the B.Tech degree in electrical engineering from the Indian Institute of Technology, and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley.

He joined the University of Houston, TX, in 1980, where he is currently a professor of computer science. His research interests are in the areas of self-stabilizing systems, inherent fault-tolerance, reliability assessment of safety-critical systems, and high performance modular parallel programming.

Dr. Bastani is currently on the editorial boards of the *IEEE Transactions on Knowledge and Data Engineering* and the Oxford University Press *High Integrity Systems Journal*. He is the editor of the newsletter of the *IEEE Technical Committee on Multi-Media Computing* and was the vice-chair of the *12th IEEE Symposium on Reliable Distributed Systems*. He was on the editorial board of the *IEEE Transactions on Software Engineering* from 1988 to 1992 and was a guest editor of the April 1992 special issue of *IEEE-TKDE* on self-organizing systems, and the December 1985, January 1986, and November 1993 special issues of *IEEE-TSE* on software reliability. He is a member of the IEEE, ACM, and INNS.



Ta-Wei Tsao received the B.S. degree in electrical engineering from the National Taiwan University in 1983, and the M.S. and Ph.D. degrees in computer science from the University of Mississippi in 1989 and 1994 respectively. His research interests are in artificial intelligence and reliability theory. Mr. Tsao is a student member of the ACM.