On Applying Imprecise Computation to Real-Time AI Systems

ING-RAY CHEN

Institute of Information Engineering, National Cheng Kung University, Tainan, Taiwan

Imprecise computation is known as a technique for real-time systems where precise outputs are traded off for timely responses to system events. This paper discusses how the technique can be applied to a class of real-time AI systems designed for solving combinatorial problems and proposes an evaluation method for assessing if imprecise computation can satisfy both the timing and functional requirements of these systems.

Received April 20 1995, revised June 28 1995

1. INTRODUCTION

Real-time computing is an open research area which represents a major challenge to engineers and computer scientists. The objective of real-time computing is to meet the timing and functional requirements of individual tasks. Thus, the most important property is predictability; the functional and timing behaviour of each task should be as deterministic as necessary to satisfy the system specifications.

The incorporation of Artificial Intelligence (AI) techniques into real-time control systems has emerged to become a state-of-the-art demand in recent years as evidenced from numerous conferences, workshops and articles [1–5] held or published each year to discuss the subject. One central theme of the discussion is how to make such AI systems real time, that is, how to ensure that the functional and timing requirements of such systems are satisfied. This issue is interesting for two reasons: (1) from the timing perspective, the exponential search time behaviour exhibited by AI programs makes them highly undesirable for real-time applications; and (2) from the functional perspective, the correctness of the output of AI programs is a fuzzy [6] rather than a binary quantity, since results produced by AI techniques may not be categorized as correct or not. For example, a non-optimal result may not be considered as completely correct because it is not the best solution; however, since a non-optimal result normally takes less time to produce than an optimal one, under a rigid time frame (e.g. minutes to seconds, such as that in managing defensive weapons against missile threats), non-optimal results may be more desirable than optimal ones because they can better satisfy the timing requirement. The existence of a real-time constraint thus complicates the design and implementation of AI real-time systems since satisfying the timing requirement may have an adverse effect on the satisfaction of the functional requirement, and vice versa. Unfortunately, current design, analysis and verification techniques for integrating AI techniques into control systems have not kept pace; little work has been done in designing and verifying the functional and timing requirements of such systems [4, 7].

Current research directions toward making AI systems real-time are conducted on an *ad hoc* basis and basically adopt one of the following two approaches. One approach is to look at parallel architecture [8–10] for better performance with real-time applications in mind in the hope that the timing requirement may be better satisfied. Another approach is to devise time-constrained search algorithms [11–14] coupled with knowledge-constrained search space [15, 16] so as to commit to actions based on limited information and computation in limited time, e.g. the result produced thus far when time expires is the one to be used since it represents the best bet. These approaches give the system designers better confidence in the embedded AI systems in control systems; however, the degree of confidence is still an open issue. Questions that remain to be answered include: (1) since fast computing does not imply real-time computing, how can one be sure that the timing requirement is always (or most of the time) satisfied using parallel architecture, and, if it does, how much confidence should the system designers have in the use of parallel architecture, (2) If time-constrained algorithms can indeed be used to satisfy the timing requirement, then how much functional requirement is compromised and, in terms of the probability that the task can be executed successfully (say, in missile systems), what is the implication of using time-constrained algorithms, i.e. how much confidence should the system designers have in these algorithms? This paper is motivated by these questions. We hope to provide implementation guidelines for implementing real-time AI systems with the assurance that both the timing and functional requirements of the system can be satisfied.

We address the issue of trading off solution quality (consequently the functional requirement is less satisfied) for guaranteed response time (consequently the timing requirement is more satisfied) in real-time AI systems by formalizing the notion of acceptability criteria under which the resulting system is considered to have satisfied both its functional and timing requirements and thus can be considered real-time. More specifically, we investigate whether the imprecise computation method [17, 18] can be applied as a specific technique for developing real-time AI systems. Under the principle of imprecise computation, more than one version of the AI system software for the same system task are developed. These versions may be implemented by incorporating time-constrained search process and/or knowledge-constrained search space, and are designed to find a solution for the same problem-solving request. However, each version in succession is given less time to produce a solution with the guarantee that some solution must be found within the specified timing constraint, although the solution found perhaps is not of the best quality. In other words, the quality of the solution is monotonically decreased as we select the first version over the second version and so on. However, guaranteed response time is assured more as we go in the same order. Of course, even with imprecise computation, some systems still cannot find a solution when time expires, especially if the time constraint is stringent. In this paper, we exclude this possibility (such that there is no deadline-violation failures) by considering a class of real-time AI systems for solving combinatorial problems for which there always exist any-time or time-constrained solutions [11], so that a solution, no matter how imperfect it can be, can always be found when time expires. Such systems include flying route-finding systems for which a direct flying route between the source and destination can be considered as an any-time solution [19], missile systems for which not considering radar threats can lead to a quick solution, and medical monitoring and caring units for which immediate imperfect treatment plans are available [4]. In these systems, lowest versions conceptually correspond to the mandatory component whose execution time must be bounded and thus can be guaranteed off line, while higher versions correspond to optional components which are to be selectively executed at run-time to refine the solution according to how much computation time remains.

To quantify the effect of trading off solution quality for guaranteed response time due to the employment of imprecise computation, we need a metric to tell whether the system, after satisfying its timing requirement this way, also satisfies its functional requirement. We define this metric the quality function of the system, which is a probability function that the functional and the timing requirements of the AI system are both satisfied for all problem solving requests encountered during the lifetime of the AI system. In this paper, we first investigate if various time-constrained algorithms can fit within the specific framework of imprecise computation, i.e. they can be used to produce various levels of solution quality under various degrees of response time requirement for the same system task. Then, after the timing requirement is satisfied this way, we propose an evaluation method to compute the quality function of the system so as to assess if the functional requirement of the resulting system is also satisfied with respect to some acceptability criteria for functionality.

The rest of the paper is organized as follows. Section 2 describes the method for implementing real-time AI systems based on imprecise computation, and discusses possible ways of implementing multiple versions of AI system software for the same system task. Section 3 presents an evaluation methodology for assessing the resulting quality function of such real-time AI systems. Section 4 illustrates the utility of the design and evaluation methodology with an example. Finally, Section 5 summarizes the paper and outlines some future research areas.

2. APPLYING IMPRECISE COMPUTATION TO REAL-TIME AI

In this section, we define the system model and discuss possible approaches for implementing real-time AI systems based on the concept of imprecise computation, with the objective of satisfying the timing requirement. In the next section, we will develop an assessment method based on the notion of acceptability criteria to quantify the trade off between the sacrifice in solution quality and the guarantee in response time.

2.1. System model

In embedded control systems, the AI system is only a part of a larger system such as a missile system. The AI system usually must provide control functions and must operate in real time in response to problem solving requests to cope with various deadlines. Our system model follows one possible structure of imprecise computation in which the AI system software can have n versions $V_1, V_2, \ldots, V_n$ for solving problem requests ($n = 2$ is the common practice). V_1 is the highest version which can presumably produce the best possible solution but may need a longer time to run, while V_n is the lowest version which may run a non-optimal, time-constrained algorithm but is able to generate a solution much quicker. Thus, the versions are ordered according to the efficiency with which they are able to produce solutions. This structure is similar in concept to that of recovery block in software fault tolerance [20].

Let $F_1, F_2, \ldots, F_n$ be the respective response time distributions and $W_1, W_2, \ldots, W_n$ (in monotonically decreasing order) be the worst-case computation times which are obtained by testing each version with the anticipated operational profile which the system is expected to encounter during its operational phase [21, 22]. In responding to a problem solving request with a deadline of t_R, the system adopts the following policy to ensure that the timing requirement is satisfied while it tries to meet the functional requirement as much as possible. The system first selects the highest version i which has a worst-case response time W_i not higher than t_R to ensure that the timing requirement is satisfied in real time with respect to the request. Since the selection of version i described above is based on the worst-case planning time, it is likely that the actual time needed to solve the problem request by version i is much less than t_R. If this is the case, the system uses the remaining time to further improve the solution quaility as much as possible, possibly by running the next higher version $i-1$ until t_R expires. This situation applies to the case when version i is implemented with an any-time algorithm. On the other hand, if version i is implemented with a time-constrained algorithm, then W_i can be set to t_R so that version i is required to generate a solution at or before t_R. The bottom line is that the real-time requirement must be satisfied from the system's perspective, although the solution quality may be compromised. Below we describe possible ways of implementing multiple versions of the AI system software to achieve such a guarantee.

2.2. Possible ways of implementing real-time AI systems based on imprecise computation

For combinatorial AI search problems, existing any-time and time-constrained algorithms (e.g. RTA^* [14], *DYNORAII* [13] and TCA^* [12]) can be used to implement lower versions while optimal algorithms (e.g. A^* [23] and IDA^* [24]) can be used to implement higher versions. For rule-based production systems [9, 25] different versions can be implemented based on imprecise computation by restricting the knowledge or information used by the AI software in searching for a solution. A real-time rule-based production system repeatedly executes the so called match-select-act cycle in which it responds to an external event (for example, a sensor event which inputs facts) by first matching arriving facts against the left-hand-side (l.h.s.) condition elements of the rules comprising the system (called the match phase), then selecting a rule to fire among the rules that are instantiated (called the select phase), and finally executing the right-hand-side (r.h.s.) actions of the selected rule (called the act phase). Firing a rule may 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 system is said to have reached a decision and the sequence of rules fired on the solution path is referred to as the solution found in response to the problem-solving request.

There are two possible ways of implementing imprecise computation in rule-based production systems. One way is to build several versions of the rule base, with higher versions being more restricted than lower versions. In other words, the algorithms used for matching, selecting and firing rules remain the same, but the rule base consists of less informative (and thus better summarized) and more constrained sets of rules as we go from higher versions to lower versions. Less informative rules can be created by grouping several rules together into one rule which summarizes the knowledge of several rules. More constrained rules can be created by not using the full expressive power provided by the rule language as the rule base is being created. These approaches reduce the matching time performed by the underlying matching algorithm during the match phase because the rule base is simplier, smaller and less powerful [15, 16]. Consequently, less time is needed to find a solution.

The second way to implement imprecise computation is to keep the rule-base the same, but use multiple versions of the algorithms used in the match, select, and/or act phases in order to reduce the planning time. For the match algorithm, lower versions can have a more restricted way of performing the match, including limiting the number of matches for a join operation and/or the number of instances of a pattern or a relation embodied in the l.h.s. condition elements of rules (as suggested in [26]). For the select algorithms, higher versions can use optimal algorithms such as A^* and lower versions can use time-constrained algorithms such as RTA^* [14] or *DYNORAII* [13] to speed up execution. For the firing phase, parallel rule firing [9] can also be considered for implementing lower versions, while sequential rule firing can be used for implementing higher versions.

There are some important points that should be mentioned. First, it is possible to combine the two approaches to implement the lower versions of the AI system software so as to reduce the worst-case computation time. Second, to guarantee that the timing requirements for all problem-solving requests are satisfied, it is necessary to perform a statistical analysis of the implemented versions to obtain the worst-case upper bound on the response time for each version. After the analysis is done, if no version (among the implemented versions) exists to satisfy the timing requirement, more restrictive rule base and/or algorithms should be sought to implement (at least) the lowest version. Third, conceptually the lowest version functions as the mandatory part of the imprecise computation process and therefore its response time must be bounded and thus guaranteed off-line. If due to non-convergence of AI techniques employed its worst-case response time cannot be bounded, we must instead implement (at least) the lowest version using an any-time or time-constrained algorithm so as to guarantee a bounded worst-case response time. In this case, it suffices to use the average rather than the worst-case response time to characterize higher versions (corresponding to the optional components) since higher versions can gradually refine the solutions based on the solutions found by lower versions when time is available. In the simplest form, the lowest version can be just a table-lookup module, listing approximate or crude solutions for some problem-solving requests under perceived conditions. Also, for a class of applications (e.g. a flight system with radar threat), it is always possible to find an immediate solution (e.g. a direct route between the source and the target without radar consideration), based on which the solution can be gradually refined when more time is available. For this latter class of real-time applications, the lowest version can be implemented by using any-time algorithms [11] to guarantee that the timing requirement is satisfied.

3. ASSESSMENT METRIC AND METHODOLOGY

In order for the system designer to have a concrete idea on whether the functional requirement has been compromised by the deployment of imprecise computation, we propose the notion of acceptability criteria which precisely define the belief of the system designer regarding a functional failure. Based on these acceptability criteria, the quality function of the embedded AI system software can be defined and later assessed based on testing of the resulting system. In our earlier work, this quality function metric was defined as the probability that the AI system can satisfy both its timing and functional requirements as a function of the number of problem requests (or missions) which the system may encounter during its life time [21]. Of course, for continuous, reactive systems, the number of problem requests which the system may encounter during its lifetime is infinity. This quality function metric in this paper now transforms into the probability function that the AI system can satisfy its functional requirement, given that the timing requirement is always satisfied due to the employment of imprecise computation.

The acceptability criteria are related to this transformed quality function metric by defining exactly how the system designer views the functional requirement has been satisfied or compromised.

In the following, we first discuss a few possible ways of defining these acceptability criteria and their relationships to the quality function metric and then we discuss a possible testing methodology with which the quality function of the embedded real-time AI system incorporating imprecise computation can be estimated from the testing result.

3.1. Acceptability criteria

We first note that a problem solving request can always meet its timing requirement due to the deployment of the imprecise computation technique. However, the functional requirement may be compromised. For example, a straight-line route for a flight system in a radar threat environment is apparently not a good solution, but it takes little time to compute. For each problem solving request, the quality of the solution generated can be considered as a random variable in the range of [0,1] with 0 meaning that the solution is totally functionally acceptable, and 1 meaning that it is totally functionally unacceptable. This assessment of the solution quality for each problem request can be done by the tester during the testing phase and conveys the belief of the system designer for the application in question. A natural way of assessing the solution quality in this way frequently exists. For example, in a flight system with radar threats, the value assignment corresponds to the probability of the flight being detected by the radar when following the solution route planned. We shall call such value in [0,1] as the ‘imperfect solution level’ (ISL).

The following acceptability criteria for functionality can be applied to real-time AI systems implemented with the imprecise computation technique.

- *Strict.* With the strict criterion, the system is considered functionally unacceptable if the system has encountered a problem solving request for which the ISL measure is above an application-specified threshold value. This criterion defines a system that cannot tolerate even a bad solution; the threshold value defines the way a system designer (or a user) views the quality level below which the system can still tolerate an imperfect solution. For example, a radar detection probability of over 0.5 may be considered functionally unacceptable for some flight systems.
- *Accumulation.* With the accumulation criterion, the system is considered unacceptable for functionality if the sum of the ISL measures of all problem-solving requests encountered by the system exceeds an application-specified threshold value, say 1. This criterion defines a system in which a single or even several bad solutions may not immediately cause the system to violate its functional requirement, but the effect may accumulate and cause the system to become functionally unacceptable. For example, a radar detection probability of 0.1 may not cause a flight system to be shot down for a single mission, but chances are if there are many solutions with non-zero radar detection probability then eventually the flight system will fail. Note that this criterion applies to systems designed to solve more than one problem request, which is typically the case for reactive real-time systems.
- *Accumulation within a mission window.* With this criterion, the system is considered acceptable for functionality as long as the accumulated ISLs encountered in an application-specified mission window do not exceed an application-specified threshold value. A ‘mission window’ means a moving window of missions, e.g. *m* means a moving window of *m* missions within which the accumulated ISLs cannot exceed a threshold limit. This criterion defines a system that can tolerate occasional imperfect solutions so long as not too many such imperfect solutions occur in any mission window. A smaller mission window hence implies a system with a better recoverability because of the lower probability of accumulating imperfect solutions’ ISLs within a smaller window to exceed a threshold, given that imperfect solutions occur at about the same rate regardless of the size of mission window. This criterion is appropriate for some real-time applications for which imperfect solutions can be digested or tolerated as long as not all of them occur in the same mission window period. For example, in a manufacturing system, under a tight production rate requirement, if too many bad products are produced in the same window period, then the functional requirement is considered not satisfied.

3.2. Relationship between acceptability criteria and quality function

The quality function of real-time AI systems implemented based on imprecise computation is driven by the acceptability criteria defined by the system designer. In this section, we analyse their relationship by using probability modelling.

With the strict criterion, we consider the system functionally unacceptable if it ever generates an imperfect solution with its ISL measure greater than a specified threshold value, σ, which defines the tolerance level of the resulting system with respect to a functionally imperfect solution. The extreme case is that when σ is 0, the system cannot withstand even a slightly imperfect solution. One possible way to obtain the expression for the quality function of the system is to model the ISL measure of an imperfect solution by a distribution $G(\cdot)$ such that $G(0) = 0$ and $G(1) = 1$, and the arrival of imperfect solutions by a Poisson process with an arrival rate λ. Let X_i be a random variable indicating the ISL of the i^{th} imperfect solution. Then, since $0 \leqslant \sigma < 1$, the quality function of the system after the system has serviced $\mathcal{N}$ requests, denoted by $Q(\mathcal{N})$, is given by

$$\begin{aligned}
Q(\mathcal{N}) &= \Pr\{\text{software is alive after } \mathcal{N} \text{ problem requests}\}\\
&= \Pr\{\text{the ISL of every imperfect solution encountered} \leqslant \sigma{<}1\text{, if any }\}\\
&= \sum_{n=0}^{\infty} \Pr\{n \text{ imperfect solutions experienced over } \mathcal{N} \text{ problem-solving requests}\}\\
&\quad \times \Pr\{X_1 \leqslant \sigma < 1, \ldots, X_n \leqslant \sigma < 1\} \qquad (1)\\
&= \sum_{n=0}^{\infty} \frac{e^{-\lambda\mathcal{N}}(\lambda\mathcal{N})^n}{n!}[G(\sigma)]^n\\
&= e^{-\lambda(1-G(\sigma))\mathcal{N}}
\end{aligned}$$

where n is the total number of imperfect solutions which can probabilistically occur in $\mathcal{N}$ problem-solving requests.

Equation (1) above gives the quality function as a function of the number of problem-solving requests for an AI system incorporating imprecise computation and adopting the *strict* acceptability criterion as its functional requirement. Naturally, if the system adopts another acceptability criterion (say, the *accumulation* criterion) for its functional requirement, the quality function expression would be different since the underlying acceptability criteria are different. For the *accumulation* acceptability criterion, if the threshold is X_L, then

$$\begin{aligned}
Q(\mathcal{N}) &= P\{\text{accumulated ISL levels experienced} \leqslant X_L\}\\
&= \sum_{n=0}^{\infty} \Pr\{n \text{ imperfect outputs encountered over } \mathcal{N} \text{ missions}\} \Pr\{X_1 + \ldots + X_n \leqslant X_L\}\\
&= \sum_{n=0}^{\infty} \frac{e^{-\lambda\mathcal{N}}(\lambda\mathcal{N})^n}{n!} G^{(n)}(X_L) \qquad (2)
\end{aligned}$$

where n is the total number of imperfect outputs 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 imperfect solutions of $G(\cdot)$ is less than x. It is defined as

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

On the other hand, if the *accumulation within a mission window* acceptability criterion is considered for which the mission window is m (missions), it can be shown that

$$Q(\mathcal{N}) = e^{-\lambda\mathcal{N}\left(1-\sum_{n=0}^{\infty}\frac{e^{-\lambda m}(\lambda m)^n}{n!}G^{(n+1)}(\delta)\right)} \qquad (3)$$

3.3. Evaluation methodology

Our evaluation methodology has its origin from the field of software reliability engineering for assessing the system reliability of computer software. Under the evaluation methodology, the system developed is tested based on its operational profile [27], from which testing results are collected so as to parameterize (i.e. give parameter values to) a quality function equation (such as Equation (1) derived based on the strict acceptability criterion) to measure the quality function of the system.

Two sets of testing data are required in order to estimate the parameters of a quality function equation such as Equation (1). These are $(N_1, N_2, \ldots, N_r)$ (for imperfect solutions) and $(f_1, f_2, \ldots, f_r)$ (for associated ISLs), where N_i is the problem-solving request number for which the i^{th} imperfect solution is found; and f_i is the ISL of the i^{th} imperfect solution. These testing data may be obtained during the testing and debugging phase through testing the AI system incorporating imprecise computation with its anticipated problem-solving request profile.

The maximum likelihood estimates (MLEs) [22] of $G(\cdot)$ and λ can be derived as follows.

The probability density of imperfect solutions is

$$PDF_{hi}(\mathcal{N}) = \lambda e^{-\lambda\mathcal{N}}$$

Therefore, the maximum likelihood estimate of λ can be estimated as

$$\hat{\lambda} = \frac{r}{N_r} \qquad (4)$$

where r is the total number of imperfect solutions experienced during the testing period, and N_r, defined as before, is the problem-solving request number for which the r^{th} imperfect solution (i.e. its $ISL > 0$) is experienced. For example, if (#5, #78, #256 #655 #1000) are a set of five problem-solving requests for which imperfect solutions are detected during the testing phase, then λ is calculated as 5/1000. In other words, the system can experience an imperfect solution once in about every 200 problem-solving requests when the system is in its operational phase.

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

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

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

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

$$\frac{r\dfrac{\partial\Gamma(\hat{\alpha}+\hat{\beta})}{\partial\hat{\beta}}}{\Gamma(\hat{\alpha}+\hat{\beta})} - \frac{r\dfrac{\partial\Gamma(\hat{\beta})}{\partial\hat{\beta}}}{\Gamma(\hat{\alpha})} + \sum_{i=1}^{r}\log(1-f_i) = 0 \qquad (5)$$

where

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

After the MLEs of λ and $G(\cdot)$ are obtained as described above, the quality function of a real-time AI system based on imprecise computation and a pre-specified acceptability criterion (e.g. such as Equation (1) based on the strict acceptability criterion) can then be quantified by using the testing data collected as a function of the number of problem requests encountered by the system.

4. EXAMPLE

This section shows an illustrative example. Consider a real-time AI planning subsystem embedded within an intelligent missile launching system that launches missiles against anti-missile threats [19]. The aim of the mission for each missile launched is to hit the target without being shot down. From the view of the missile, the sky (from some particular altitude looking down) is a two-dimensional *x–y* map with certain locations marked with anti-missile threats and associated intensities. As the missile's altitude/location changes as it moves toward the target, the corresponding *x–y* map changes, thereby creating a map by map three-dimensional search space through which the AI program needs to find a best flying route for each missile launched to accomplish its mission. We consider the case that the AI program is implemented with two versions based on imprecise computation. Both versions must consider the physical constraints of the missile dynamics, e.g. no backward, and sudden vertical movements, etc. The first version uses an optimal search algorithm called A^* [23] which, when given sufficient time, can always find the best flying route among all in terms of the smallest probability of being shot down. It considers the whole search space as it looks for the optimal route. The second version, on the other hand, uses a suboptimal search algorithm called RTA^* [14] coupled with an any-time algorithm [11]. Under the second version, the missile moves toward the target in increment of horizontal distance window (e.g. 50 kilometres) nearer to the target one at a time such that within each distance window the probability of being shot down is the minimum. (See Figure 1 for an illustration of the *x–y* map window.) In other words, the search space is only one distance window at a time (at lower heights as it approaches the target) instead of the whole distance spanning the source and the target as having been done by the first version. Furthermore, in order to guarantee a timely response for the missile launching system to make a decision to launch a missile at or before the deadline t_R, the second version will use the straight-line route between the end distance point planned so far and the target point as its last part of the flying route when t_R expires. By this way, the second version will always find some flying route to reach the target, although the probability of being shot down against the anti-missile threats is perhaps not the smallest.

We consider that whenever the missile launching system is ready to launch a missile, it can obtain real-time information regarding the anti-missile distributions and intensities and also a deadline t_R, both of which vary on a mission by mission basis depending on the real-time situations. The worst case planning time of the first version is determined a priori by testing it through a simulated environment profile (consisting of the anti-missile distributions and intensities, and target location distributions). On the other hand, since the second version always finds a flying route with any deadline t_R, no simulation experiment is necessary for the second version. The missile launching system behaves as follows: for a given t_R, it selects the first version over the second version if the worst case planning time of the first version is shorter than t_R given; otherwise, the second version is chosen. In the latter case, if the second version finds a route before t_R, the first version is invoked using the remaining time; if a route is found by the first version before time expires, the route found by the first version replaces the one found by the second version since it has a smaller probability of being shot

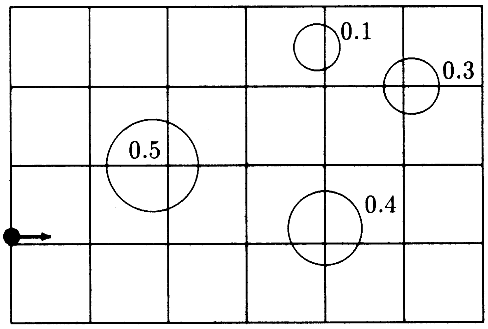

FIGURE 1. An *x–y* map window marked with anti-missile threats and intensities.

[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).

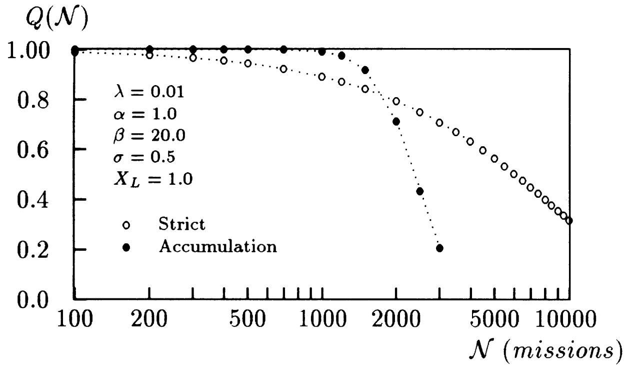

FIGURE 2. Quality function of the missile launching system with strict and accumulation criteria.

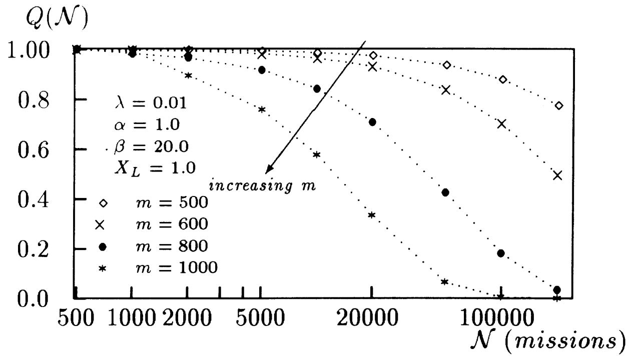

FIGURE 3. Quality function of the missile launching system with the accumulation within a mission window criterion.

down. In any case, the route found is used by the launching system to control the actual flying route of the missile launched so as to minimum the possibility of being shot down.

The designer now wishes to know what the quality function of the system looks like with such a design based on imprecise computation. This question is equivalent to knowing whether the system can satisfy both the timing (already satisfied via imprecise computation) and functional requirements. To find the answer, the system implemented was tested through its simulated operational profile (consisting of the environment profile plus the distribution of t_R) one mission at a time and history data were collected which consisted of $(N_1, N_2, \ldots, N_r)$ (for imperfect solutions) as well as the associated $(f_1, f_2, \ldots, f_r)$ (for ISLs), with $f_i > 0$ denoting the non-zero probability of being (detected and) shot down by the anti-missile threats. The data were subsequently used to compute the values of the model parameters based on the method discussed in Section 3.3, yielding $\lambda = 0.01$; $\alpha = 1.0$ and $\beta = 20.0$.

As the quality function of the system must reflect the imposed functional requirement demanded by the system, the system designer tests three acceptability criteria deemed appropriate for the system under evaluation. Figure 2 shows the evaluation result in which the quality function is expressed as a function of the number of missions (missiles launched by the launching system) for two acceptability criteria: strict with $\sigma = 0.5$ and accumulation with $X_L = 1.0$. It can be seen that the acceptability criterion selected affects the quality of service delivered by the system. The quality function of the system with the accumulation criterion with $X_L = 1.0$ deteriorates quickly after an upper bound of 1000 is exceeded. If the system designer believes that the accumulation criterion with $X_L = 1.0$ is the right choice, it is better that the system will only launch 1000 missiles or less to the same target area or the functionality of the system is in great risk. If we assume that missiles are launched to the same target area in one batch job, then the system can satisfy its imposed functional requirement (with probability near 1) with the batch size less than 1000.

On the other hand, if the system designer believes that the strict criterion with the tolerance threshold $\sigma = 0.5$ should be adopted, then the functionality steadily deteriorates as the system launches more and more missiles. The probability that the system can satisfy its functional requirement in this case drops to 0.9 after launching 750 missiles. This result is reasonable because the strict criterion requires that every missile launched must have its probability of being shot down to be less than 0.5 or the system is considered as having violated its functional requirement. Figure 2 also shows the cross-over point (at around 2000) beyond which the strict criterion is better than the accumulation criterion in terms of meeting the functional requirement.

Figure 3 shows the quality function of the system with the accumulation within a mission window criterion for which the system's tolerance to imperfect results is modelled by a mission window m. The smaller the value of m, the better the system's ability to tolerate occasional bad results. At one extreme, where $m = \infty$, a system with the accumulation within a window criterion behaves just like a system with the accumulation criterion alone because the system fails as soon as the accumulated ISLs exceed $X_L = 1$. At the other extreme, $m = 0$ represents that the system is able to tolerate bad results instantaneously, in which case the quality function is always 1. For any m value between these two extremes, as m increases the value of the quality function decreases for the same number of missions, because a larger m implies a higher probability that the accumulated ISLs in the mission window m may exceed X_L, thereby making the system more vulnerable to imperfect solutions. Note that m is to be specified by the system designer. For this example, Figure 3 suggests that when $m = 500$ or 600 the system's functional requirement can be satisfied with a high probability (close to 1) for up to 10 000 missiles launched.

The quality function curves obtained in Figures 2 and 3 thus provide the system designer a firm idea about how the functionality of the system is compromised as a result of adopting imprecise computation to trade off solution quality for guaranteed response time. It is important to note that the quality function curve varies as the acceptability criterion chosen by the system designer varies. For example, if the threshold σ value is 0.75, the whole curve under the strict criterion in Figure 2 will move up toward 1. If the system designer is not satisfied with the shape of the curve, he or she will have to redesign and re-evaluate the AI programs for one or even all versions because it is of little value for the system to satisfy the timing requirement at the entire expense of the functional requirement.

5. SUMMARY

Imprecise computation is a technique suitable for real-time systems for which a response must be generated within a real-time deadline or catastrophe may result. In this paper, we discussed how imprecise computation can be applied to implementing AI programs embedded within real-time systems. We proposed the notion of acceptability criteria for functionality to quantify the trade-off between the sacrifice in solution quality and the guarantee in response time. For a chosen acceptability criterion as deemed appropriate for the system under evaluation, we developed a method for quantifying the system's functional requirement, expressed in terms of the probability of satisfying the acceptability criterion as a function of the number of missions serviced by the system during its lifetime. Analytical expressions for this system quality function with various acceptability criteria were derived and a detailed example was shown to demonstrate the utility of the result. The methodology developed in the paper is invaluable for system designers who wish to apply imprecise computation to building real-time AI systems; it allows the system designer to evaluate whether an implemented system can indeed satisfy both the timing and functional requirements of the system.

A possible future research area is to apply the method to the design, development and evaluation of real-time rule-based programs embedded in process-control systems.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Council, Republic of China, under Grant NSC 85-2213-E006-069.

REFERENCES

[1] *10th IEEE Conference on AI for applications*, San Antonio, March, 1994.

[2] *2nd International Conference on AI Planning Systems*, Chicago, June 1994.

[3] *3rd IFAC International Workshop on AI in Real Time Control*, Sep. 1991, Sonoma Valley, Calif., USA.

[4] D. J. Musliner, et al., 'The challenges of real-time AI,' *IEEE Computer*, Jan. 1995, pp. 58–66.

[5] *IEEE Transactions on Knowledge and Data Engineering*, Special issue for Dependability of AI programs, Feb 1995.

[6] L. A. Zadeh, 'Fuzzy sets and information granularity,' *Advances in Fuzzy Set Theory and Application*, North-Hollard, 1979.

[7] J. A. Stankovic, 'Misconceptions about real-time computing: a serious problem for next-generation systems,' *IEEE Computer*, Vol. 21, No. 10, Oct. 1988.

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

[9] T. Ishida, 'An optimization algorithm for production systems,' *IEEE Trans. on Knowledge and Data Engineering*, Aug. 1994, pp. 549–558.

[10] 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, March 1991, pp. 3–10.

[11] M. Boddy and T. Dean, 'Solving time-dependent planning problems,' *11th International Joint Conf. on AI*, 1989, pp. 979–984.

[12] L. C. Chu and B. W. Wah, 'Solution of constrained optimization problems in limited time,' *IEEE Workshop on Imprecise and Approximate Computation*, 1992, pp. 40–44.

[13] B. Hamidzadeh, and S. Shekhar, 'Specification and analysis of real-time problem solvers,' *IEEE Trans. Soft. Eng.*, August 1993, pp. 788–803.

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

[15] C. J. Paul, A. Acharya, B. Black, and J. K. Strosnider, 'Reducing problem-solving variance to improve predictability,' *Communications of the ACM*, August 1991, pp. 80–93.

[16] J. K. Strosnider and C. J. Paul, 'A structured view of real-time problem solving,' *AI Magazine*, Summer 1994, pp. 45–66.

[17] J. Liu, K. J. Lin, W. K. Shih, and A. C. Yu, 'Algorithms for scheduling imprecise computation,' *IEEE Computer*, May 1991, pp. 58–68.

[18] W. L. Shin and Jane W. S. Liu, 'Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error,' *IEEE Trans. Computers,*, Vol. 44, No. 3, March 1995, pp. 466–471.

[19] 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, 1993, pp. 570–577.

[20] B. Randell, 'System structure for software fault tolerance,' *IEEE Trans. Soft. Eng.*, Vol. 1, No. 1, June 1975, pp. 220–232.

[21] I. R. Chen, F. B. Bastani and T. Tsao, 'On the reliability of AI planning software in real-time applications,' *IEEE Trans. Knowledge and Data Eng.*, Feb. 1995, pp. 4–13.

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

[23] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[24] R. E. Korf, 'Depth-first iterative deepening: an optimal admissible tree search,' *Artificial Intelligence*, Vol. 27, 1985, pp. 97–109.

[25] I. R. Chen and T. Tsao, ‘A reliability model for real-time expert systems,’ *IEEE Trans. Reliability*, March, 1995, pp.54–62.

[26] P. N. Haley, ‘Real-time for RETE,’ *3rd Workshop on Robotics and Expert Systems*, 1987.

[27] J. D. Musa, ‘Operational profiles in software reliability engineering,’ *IEEE Software*, March 1993, pp. 14–32.