An Approach to Quantifying Tradeoffs in Testing Intelligent Systems

Deborah Walker and Piotr Gmytrasiewicz

Computer Science and Engineering Department
University of Texas at Arlington
Arlington, TX 76019
dwalker@metronet.com, piotr@cse.uta.edu

Abstract

The work we are presenting extends the scenario based testing methodology to make it applicable to the testing of intelligent software systems. Due to the unpredictability of the environments in which intelligent systems are deployed, we have extended the scenario tree concept to include equivalent classes of events and system states, and the probability with which the classes of events are expected to occur. Using the notion of utility, we define the concept of importance of testing an event in an intelligent system, possibly equipped with a learning module. Finally, we quantify the degree of confidence achieved by partial testing of a system. Our approach allows the designer to determine which test cases should be executed to meet the given confidence requirements, and to examine the tradeoffs between further testing and increased confidence in the system.

1. Introduction

As intelligent systems become more and more prevalent in today's software applications, the need to ensure their quality and reliability becomes imperative. Software test engineers are faced with a unique task when testing intelligent systems, since traditional software engineering techniques are not directly suitable for intelligent applications. Traditional software systems can simply be tested using any accepted method developed to test software with static processing capabilities. Intelligent software, however, has both static and dynamic processing capabilities—that is, it processes data differently over time. The need to differentiate these two properties of intelligent systems has been uncovered before (Preece, Grossner, & Radhakrishnan, 1996). A detailed analysis of the specific dynamic properties of an intelligent system is beneficial in white-box testing techniques, as shown by (Chang, Combs, & Stachowitz 1990; Evertsz 1992; Preece, Shinghal, & Batarekh 1992). However, it is often the case that a white-box testing approach is unsuitable for certain situations, and a black-box testing approach is desired. This involves merging the static and dynamic portions so the system may be evaluated as a whole. This is not license to test the system in a conventional manner; the dynamic aspects of the system must still be accounted for. However, as noted previously by (Rushby 1988), the opportunity is available to adapt conventional testing methods to the realm of intelligent systems. The concepts illustrated in this paper take advantage of one such opportunity. We have adapted the scenario based approach to create a new model for intelligent systems testing strategy and have extended this model to quantify the tradeoffs between testing efforts and the confidence in the intelligent system's reliability.

2. Scenario Based Testing Approach

One commonly used method to develop test cases for traditional, repeatable, software systems is to use the scenario based approach. Scenario testing involves identifying every possible sequence of events encountered by a system, then testing each scenario, or sequence, independently. The scenarios form a tree, as illustrated in Figure 1. In Figure 1, the root node, Idle, represents the initial state of the system. Every possible event that can cause a transition out of this state results in the tree branching down from this initial state. Every complete path from root to leaf represents a scenario. Although in reality any number of cycles may exists between nodes in the same scenario, such cycles are omitted from the model since testing the cycles would result in duplicated effort.

In the traditional approach to scenario testing, one assumes that every leaf node is the same as the root. This is because in a well-behaved traditional, and repeatable, system, each sequence of events will inevitably return the system to its initial state. To apply scenario based testing to the system represented by this scenario tree, test cases are chosen so that every scenario is tested—that is, every edge and node are traversed by the test cases at least once. This testing method is exhaustive and ensures complete test coverage.

Intelligent systems and the environments in which they operate are inherently different from traditional systems. First, a traditional software system has a finite number of states, while an intelligent software system may have an infinite number of states. This is due to the fact that, unlike traditional software, intelligent systems can learn, i.e., increase their knowledge over time by adding information to their knowledge base, and adapt their behavior to suit the environment they encounter. This

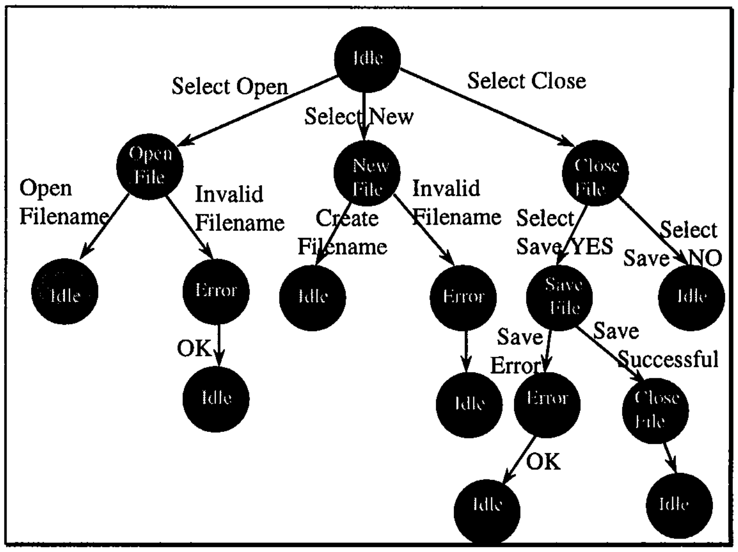

Figure 1: Scenario Tree Model of a Traditional Software System (a simple text editor).

constant adaptive process prevents an intelligent system from returning to its own previous, less informed states. Furthermore, the requirements and operating constraints of a traditional system are generally concrete and enumerated during the requirements analysis. Thus, the events that can occur are knowable and can be simulated by test cases. An intelligent system is usually required to operate in an environment that is normally not fully known beforehand. The number and exact nature of some of the events that may occur are therefore uncertain.

It is for these two reasons that traditional software testing methods are not directly suitable for intelligent systems. Since the number of events is unknown and possibly infinite, and the system states are non-repeating, the scenario tree has infinite breadth and depth. Thus, exhaustive testing of an intelligent system using the scenario based approach is not possible because the size of the scenario tree explodes beyond testable bounds.

In an attempt to fill this void, we have formalized an approach that modifies scenario based testing to make it applicable to intelligent systems. We have enriched the scenario tree model with new features to provide a systematic and customizable testing strategy that quantifies the increase of confidence resulting from more extensive testing. In Section 3 we present a modified scenario tree modeling technique applicable to intelligent systems. In Section 4 we outline the procedure required to use this model for testing purposes, concentrating on the evaluation of confidence and exploring the tradeoffs between complete test coverage and required test efforts. We conclude by listing possible extensions of the conceptual model we have described.

3. Adapting the Scenario Tree Model

The scenario tree model, with some straightforward modifications, provides an appropriate representation of an intelligent system's interaction with its environment. Specific adjustments are necessary to account for the unique nature of an intelligent system. Out modifications follow the spirit of using Markov models for statistical based testing (Feinberg & Shwartz, 1991).

First, to compensate for the infinite number of external events that could potentially arise in an uncertain domain, we use the concept of an event class. Rather than enumerating each individual event and allowing it separate treatment, similar events are grouped into abstract event classes. Although the most appropriate method used to identify event classes will vary greatly between specific applications, we require that all events in an event class be equivalent in that they invoke similar reactions by the system, and result in equivalent system states. In other words, our concept of the event class is based on the assumption that the resulting states of the system are indistinguishable from one another for testing purposes. Thus, validating the system for any one event in a class ensures that all other member events have also been tested. To account for the unexpected events, we introduce an additional class with unknown members. This class can not be tested, but its inclusion is vital for evaluating the achieved confidence. The event class concept keeps the number of events in the scenario tree model within reasonable limits.

Second, to further specify the nature of our event class labeled edges, we require that probabilities be assigned to each. Thus, every event class branching down

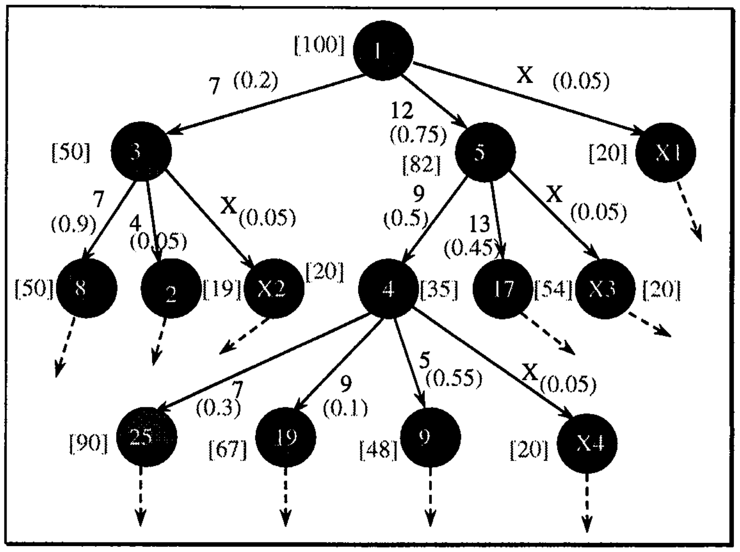

Figure 2: Scenario Tree Model Adapted to an Intelligent System.

from a node in the scenario tree will be assigned the probability of the occurrence of an event from that event class. Domain knowledge or statistical data should serve as a guide when assigning probabilities. We use the notation P_n, to indicate the probability of an event which transitions the system into state n.

Before elaborating further, we should present an abstract representation of an intelligent system state. A traditional system state can be identified solely by all the event-next-state pairs that branch down from a state. We will call this the static portion of the system state. An intelligent system, in addition to the static portion of its state, also has a dynamic portion, which is associated with the system's evolving knowledge base. When examining the scenario tree model, although the knowledge base state of the system is ever changing, we will label the nodes with the same static portions as related states, differentiated only by what the system had learned in the interim. A node with a static portion previously encountered in the same scenario will be referred to an intermediate leaf. We introduce the concept of state with static and dynamic portions to allow our model to provide further specification for intelligent systems with learning capabilities. Regardless of the particular learning method, such systems, when faced with previously encountered environmental conditions, should be able to perform better based on what was previously learned. Our method allows us to incorporate this fact into the confidence measure of the tested system.

Finally, we introduce into our model the concept of utility. We follow the utilitarian paradigm of intelligent system design advocated, for example, by (Russell & Norvig, 1995). According to this paradigm, an intelligent system is equipped with a measure, called utility, describing the degree to which the system has fulfilled its objectives. The behavior of an intelligent system should be such that utility is maximized. Therefore, we assign a utility value, U_n, to every state, n, in our model, that represents the degree to which the system's objectives are fulfilled in the state resulting from the last state transition. This value can be computed by using the multiattribute utility theory (see Russell & Norvig, 1995, and the references therein).

Figure 2 depicts the new scenario tree model we have produced by incorporating these new concepts. Utility values are shown as $[U_n]$ and probabilities are shown as (P_n).

4. Evaluating Testing Strategy

Now that we have a scenario tree model that is manageable in size and that has identifiable event patterns, we can present a procedure that evaluates the confidence produced by a partially tested scenario tree, and that adds new test cases so as to maximize the incremental increase in confidence. First, we introduce the concept of an importance factor that will be crucial to this procedure. Every node, n, has an importance factor, IF_n, which is an indication of the importance of testing the event class that causes the transition to this node, given that all of the preceding events in this scenario have already been tested. A similar concept has been explored previously by (Miller 1990), who suggested that the most likely faults should be considered the most important to test. In our method, the importance factor, IF_n, is a function of two variables. The first variable is related to the utility value, U_n. Similarly to (Miller 1990), we postulate that a state transition that causes a dramatic shift in the utility value warrants testing. We measure this effect by considering the absolute value of the difference between the utility of the node and the utility of the node's parent. In this way we promote testing of scenario paths with either disastrous or promising consequences and discourage testing of event sequences that are inconsequential and do not affect the system's utility.

If the system in question is not provided with a learning mechanism, then the change in utility alone determines the node's importance factor. However, given a system that does learn, we use the intermediate leaf concept to determine the path length between nodes with repeated static states to modify the node's importance. Path length gives an estimate of how many events have occurred since the system was last in the same static state. We postulate that as the number of intermediate events increases, i.e., the path length grows, the system's dynamic portion is more likely to have drastically changed, and the importance of testing the intermediate leaf at the end of the path should increase. If a node, x, is an intermediate leaf, then the node's predecessor, y, must be located and the logical path length, PL_{xy}, between them must be calculated. Equation 1 provides the formula necessary for this computation.

$$PL_{xy} = \begin{cases} E + 1 \text{ if } z = 1 \\ E + [1 - (1 / (C - z + 1))] \text{ if } z > 1 \end{cases} \qquad (1)$$

Where E is the number of edges between x and y, C is the number of events in an event class, and z is the number of times an event class has been encountered along the path between x and y. The effect of this is to reduce the importance of testing the system's reaction to events that have been previously encountered.

Recall that IF rates the degree of importance to test a node, i, relative to it's parent node, j. For an intelligent system that does not learn, IF can be computed using only change in the utility values. Equation 2 gives the formula for the IF for any node, n, in the scenario tree of a non-learning intelligent system. Equation 3 provides the calculation given a system that does learn.

$$\text{(non-learning)} \quad IF_n = | U_i - U_j | \qquad (2)$$

$$\text{(learning)} \quad IF_n = |U_i - U_j| \times [1 - (1 / PL_{xy})] \qquad (3)$$

We found it convenient to normalize the IF factors. The normalized importance factor, NIF, represents the importance of testing a node relative not only to its parent, but also to its siblings. Equation 4 gives the formula for calculating NIF_n.

$$NIF_n = IF_n / [IF_n + \sum IF_k] \qquad \textbf{(4)}$$

Where the summation is over all siblings k of the node n. Figure 3 shows the NIF for each node in the example tree depicted in Figure 2.

Node #	NIF
3	0.338
5	0.122
X1	0.540
8	0.000
2	0.508
X2	0.492
4	0.343
17	0.204
X3	0.453
25	0.478
19	0.278
9	0.113
X4	0.130

Fig. 3
The Importance Factor
for each Tree Node.

Now that we have explained the information represented by the NIF and how it is calculated, we can show how it is used for testing purposes. Recall that due to the nature of intelligent systems and their operating environments, one can not exhaustively test the system and thus can never be guaranteed of complete test coverage. Rather, we can only have a certain degree of confidence that our test cases have adequately validated the system. The concept of a degree of confidence will thus be incorporated into our model. By applying the NIF and the probabilities, we can estimate, for each node, our increase in confidence, IC_n, achieved by encompassing that node in a test case scenario. Equation 5 provides the formula for calculating IC.

$$IC_n = NIF_n * \prod P_n \qquad \textbf{(5)}$$

The product in Equation 5 includes all probabilities along the path from the root to the node n. product of all probabilities on the path leading to the node. Figure 4 shows the IC values for each node in Figure 2.

Test cases can be chosen that test a scenario to a terminating node that produces an acceptable degree of confidence for the given application. To estimate the overall confidence, OC_N, achieved by a given partial coverage of the scenario tree, by including a set of nodes N, we first compute the IC of each node already encompassed by the test set. Here, we set the confidence of the root node to 0, since testing does not apply to the initial state of the system. Then, overall confidence is calculated as indicated in Equation 6.

Node #	confidence increase
3	0.0676
5	0.0915
X1	0.0270
8	0.0000
2	0.0051
X2	0.0049
4	0.1286
17	0.0689
X3	0.0170
25	0.0538
19	0.0104
9	0.0233
X4	0.0024

Fig.4
The Increased Confidence
for each Tree Node.

$$OC_N = \sum IC_n \text{, where } n \in N \qquad (6)$$

Overall confidence is the percentage of a software system that has been tested, and is known to be functioning properly. Working with our example tree and assuming we choose terminating nodes 2, 19 and 9, we find that N = {3, 5, 2, 4, 19, 9}. This produces an overall confidence of 32.65%.

Conclusion and Future Work

As the software world attempts to assign more and more complex tasks to computers, the need to effectively employ intelligent systems technology is increasing. The possible results of allowing an intelligent system to operate without having any indication of how well it will perform could be disastrous. We have presented a model to represent the unique characteristics of intelligent systems by adapting the widely used scenario tree model. Our model quantifies the confidence achieved by partial testing of the scenario tree. Conversely, given the desired confidence to be achieved by testing, the tree can be used to generate the minimum test cases that will achieve this confidence. The algorithm to generate the minimum test cases requires some type of look-ahead mechanism or heuristic search, and is currently being developed.

The precise steps and calculations needed to evaluate the overall confidence of a test strategy are outlined and related to our scenario tree model. The methodology is both intuitive and systematic. It is general enough to apply to a wide variety of intelligent systems, yet specific enough to ensure validity.

It is our intention to expand our analysis of this model to include algorithms that will search the scenario tree and produce an optimal testing depth value for every scenario to assure the required confidence level and minimize the number of test cases needed to achieve this confidence. We are also developing a formal proof showing that the Overall Confidence converges to 1 (100%) if the tree is tested to infinite depth along all paths.

References

Chang C. L., Combs J. B., & Stachowitz R., 1990. A report on the Expert Systems Validation Associate (EVA). *Expert Systems with Applications* 1(3):217-230.

Evertsz R., 1991. The Automatic Analysis of Rule-based System Based on their Procedural Semantics. In Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI '91*),* 22-27.: International Joint Conferences on Artificial Intelligence, Inc.

Feinberg E. A. & Shwartz A., 1991. Markov Decision Models with Weighted Discounted Criteria. Technical Report, TR 91-43, The University of Maryland.

Hsia P., Samuel J., Gao J., Kung D., Toyoshima Y., & Chen C.. 1994. Formal Approach to Scenario Analysis. IEEE Software 11(2):33-41.

Miller L. A., 1990. Dynamic Testing of Knowledge Bases Using the Heuristic Testing Approach. *Expert Systems with Applications.* 1(3):249-269.

Preece A. D., Shinghal R., & Batarekh A., 1992. Verifying expert systems: a logical framework and a practical tool. *Expert Systems with Applications.* 5(2):421-436.

Preece A. D., Grossner C., & Radhakrishnan T., 1996. Validating Dynamic Properties of Rule-Based Systems. International Journal of Human-Computer Studies. 44:145-169.

Rushby J., 1988. Quality Measures and Assurance for AI Software. Project, SRI-CSL-88-7R, SRI Project 4616.

Russell S. and Norvig P. 1995. *Artificial Intelligence: A Modern Approach.* Upper Saddle River, NJ.:Prentice Hall.