HKU CSIS Tech Report TR-2004-05

To appear in *Proceedings of the 4th International Conference on Quality Software* (*QSIC 2004*),
IEEE Computer Society Press, Los Alamitos, California (2004)

Towards the Application of Classification Techniques to Test and Identify Faults in Multimedia Systems [*] [†]

M. Y. Cheng
The University of Hong Kong
mycheng@cs.hku.hk

S. C. Cheung
Hong Kong University of Science and Technology
sccheung@cs.ust.hk

T. H. Tse [‡]
The University of Hong Kong
thtse@hku.hk

Abstract

The advances in computer and graphic technologies have led to the popular use of multimedia for information exchange. However, multimedia systems are difficult to test. A major reason is that these systems generally exhibit fuzziness in their temporal behaviors. The fuzziness is caused by the existence of non-deterministic factors in their runtime environments, such as system load and network traffic. It complicates the analysis of test results. The problem is aggravated when a test involves the synchronization of different multimedia streams as well as variations in system loading.

In this paper, we conduct an empirical study on the testing and fault-identification of multimedia systems by treating the issue as a classification problem. Typical classification techniques, including Bayesian networks, k-nearest neighbor, and neural networks, are experimented with the use of X-Smiles, an open source multimedia authoring tool supporting the Synchronized Multimedia Integration Language (*SMIL*). *The encouraging result of our study, which is based only on five attributes, shows that our proposal can achieve an accuracy of 57.6 to 79.2% in identifying the types of fault in environments where common cause variations are present. A further improvement of 7.6% is obtained via normalization.*

Keywords: *Software testing, multimedia, classification, Bayesian networks, k-nearest neighbor, neural networks*

1. Background of the Problem

The advances in computer and graphic technologies have led to the popular use of multimedia for information exchange today. Many applications, whether based on personal computers or cellular phones, are designed to deliver rich multimedia contents. It is important to ensure the quality of the multimedia systems, especially for the temporal behaviors of rich content presentations. For example, e-learning is one of the well-known multimedia applications and has become a popular trend in education owing to its flexibility. Students can learn at their own pace at their own time. If an e-learning slide show and the voice-over are not synchronized, however, students may not able to understand the course materials. This may discourage the use of e-learning.

Conventional approaches in software testing construct test cases and apply them to the implementation under test (IUT) against the expected results, that is, the test oracles [9]. Such approaches, however, may not be appropriate to the testing of multimedia systems with intrinsic fuzzy behaviors. The fuzziness in multimedia systems is often caused by uncertainty or uncontrollable factors in their environments, such as intrinsic random noises. They are known as *common cause variations* [23], which are inherent in all multimedia processes. It is, for instance, difficult to start playing several multimedia contents at exactly the same time. In the previous e-learning example, the slide show and its audio narration are supposed to be played simultaneously. Nevertheless, if we measure the start time of the slide show and that of the audio narration, we may find that they are not played exactly at the same time. There are many possible causes of variations,

[*] © 2004 IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Personal use of this material is permitted. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse anyrcopyrighted component of this work in other works must be obtained from the IEEE.

[†] This research is supported in part by a grant of the Research Grants Council of Hong Kong and a grant of The University of Hong Kong.

[‡] All correspondence should be addressed to Dr. T. H. Tse, Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong. Telephone: (+852) 2859 2183. Email: thtse@hku.hk.

such as the degrees of real-time support of the underlying hardware, the network, and the operating system.

Little work has been done on the testing of multimedia systems, especially with respect to temporal relationships. According to Ziv and Richardson [25], uncertainty is inherent and inevitable in the software development process, including the testing phase. When testing temporal relationships of multimedia systems, uncertainties may be introduced by the execution environment [26]. Hence, the testing accuracy will be affected. In our previous work [13], we have also shown that it is difficult to test temporal relationships in multimedia systems owing to many uncontrollable factors in the environments, such as network congestion, transmission delays, packet loss, and so on. We have developed a generic framework to test the timing in distributed multimedia software systems. There are, however, a few limitations in our first attempt:

(*a*) To solve the problem of common cause variations, we used the conventional technique of a *tolerance zone*, which is an acceptable range of values. Thus, users are required to specify the tolerance range of the temporal relationships under test. The resultant values within the range will be regarded as equal to the expected value. It is, however, difficult to set a suitable tolerance zone. If the tolerance range is too large, some failures may not be revealed. On the other hand, if it is too small, testers may wrongly reject acceptable systems. As a result, the subjective tolerance may affect the judgment as to whether the systems contain faults.

(*b*) We proposed to test one temporal relationship at a time. We assumed that an *n*-ary temporal relationship can be expressed in terms of multiple binary relationships to be tested individually. In the e-learning example, for instance, testers may be interested in whether the slide show and its audio narration are played over the same period of time. In this case, two temporal relationships need to be considered: One is the difference between the start times of the slide show and the audio narration, and the other is the difference between the end times of the slide show and the audio narration. When the number of media objects increases, the number of temporal relationships may increase exponentially.

(*c*) Furthermore, we could not detect faults that would only exhibit their effects when multiple relationships occur at the same time. Consider, for example, a media object with an expected start time of 0 second and an expected end time of 1 second. Suppose the tolerance is set to 2 seconds. Consider two implemented systems: A system *A*1 with a start time of 1 second and an end time of 3 seconds, and a system *A*2 with a start time of 2.1 seconds and an end time of 3.1 seconds, as shown in Figure 1. If we review the start and end times independently and compare each variation with the tolerance, system *A*1 will be accepted while system *A*2 will be rejected. However, the media playback is much more distorted in system *A*1 than that in system *A*2 — the duration of system *A*1 is actually doubled.

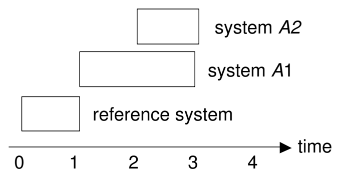

Figure 1. Time-line diagram of potentially faulty systems *A*1 and *A*2.

In this paper, we propose to consider the testing and fault-identification of multimedia systems as a classification problem. It defines the features of the systems under test using the statistical characteristics of the sample data, without the need to set tolerance limits subjectively. Furthermore, the statistical features are independent of the number of temporal relationships involved and, hence, the classification approach may also alleviate the problem described in paragraph (*c*) above. A preliminary empirical study has been carried out to investigate the feasibility of applying existing classification techniques to the situation.

The rest of this paper is organized as follows. The next section describes related work on testing multimedia systems. Section 3 describes an empirical study on the application of classification techniques to test an open source multimedia program X-Smiles that accepts SMIL documents as input. Three commonly used classification techniques (Bayesian networks, *k*-nearest neighbor, and neural networks) are described. The experimental results are reported in Section 4. In Section 5, we discuss further observations regarding the effectiveness of the classification techniques with respect to the characteristics of multimedia systems. Finally, we draw conclusions and summarize future work in Section 6.

2. Related Work

According to Blakowski and Steinmetz [10], a multimedia system should be one that can process several media types with at least one time-dependent medium. A temporal dependency among media objects is known as a *temporal relationship*. Allen [6] has proposed seven main types of temporal relationship between two media objects, as shown in Figure 2. In our previous work [13], we recognize that three temporal relationships, as shown in Figure 3, should be more fundamental. They include

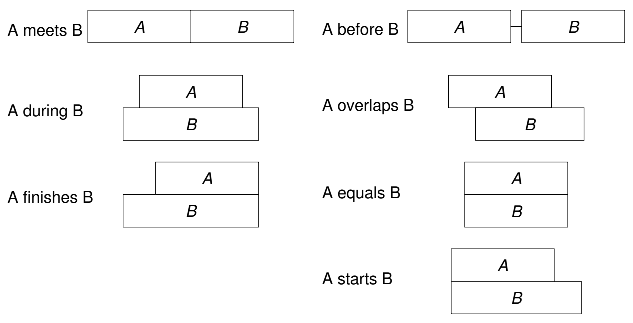

Figure 2. Temporal relationships between two media objects.

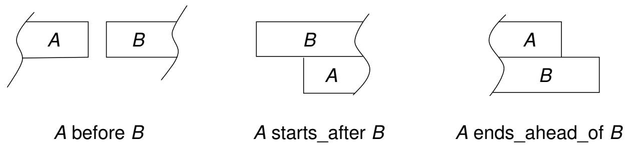

Figure 3. Fundamental temporal relationships between two media objects.

***A before B*:** which stands for media object *A* terminating before media object *B* begins;

***A starts*_*after B*:** which stands for media object *A* starting after media object *B* has begun; and

***A ends*_*ahead*_*of B*:** which stands for media object *B* terminating before media object *A* has ended.

Various researchers have developed synchronization and storage models for multimedia objects. For example, Little and Ghafoor [18] have proposed a model known as Object Composition Petri Net, which is a Petri net with time values to model synchronization in multimedia. An algorithm to retrieve the media elements from the database is proposed. They suggest a tree-based approach to store and retrieve multimedia objects.

Previous work related to the handling of uncontrollable factors in multimedia systems, such as the use of tolerance zones, has already been highlighted in Section 1 and will not be repeated here. Various researchers recommend different tolerance values for different temporal relationships. For example, the tolerance limit between an audio and a video is set as ± 80 ms in [20]. This type of suggestion, however, is rather subjective. It may not be appropriate for us to apply the same suggested value for multimedia applications in environments with different characteristics. For instance, the same value may be too conservative for desktop computers and too aggressive for cellular phones, since people are usually more tolerant about multimedia applications in cellular phones than those in desktop computers.

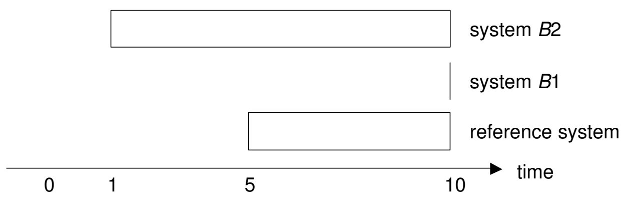

Figure 4. Time-line diagram of faulty systems *B*1 and *B*2.

Cheung, Chanson, and Xu [13] have discussed the problems of testing temporal relationships and developed a generic framework for temporal testing for distributed multimedia software systems. The paper uses a statistical approach to test temporal relationships. It proposes to consider one temporal relationship at a time. Testers are required to explicitly specify the tolerance value of the temporal relationship under test. They run the test several times and calculate the mean and sample variance. From the lookup table, one can obtain the confidence level of rejecting the implementation under test. Nevertheless, there are a couple of limitations. First, when the number of media objects increases, the number of temporal relationships may increase exponentially. It may not be feasible to consider one relationship at a time when the number is large. Secondly, the technique does not help identify the type of fault. Suppose, for example, that a faulty system *B*1 sets the value of the start and end times of a media object to be identical, while another faulty system *B*2 sets the start time of a media object to be exactly 1 second. Figure 4 shows both faulty systems with respect to the reference system, which is expected to be free from fault. Using the technique of [13], both *B*1 and *B*2 will be rejected with a high level of confidence. There is no additional information to reveal the most likely fault.

3. Applying Classification Techniques to Solve the Problem: an Empirical Study

We have conducted an empirical study on the application of classification techniques to help test and identify faults in multimedia systems. Two media objects are used in each implementation under test, involving a slide show and an audio narration. We assume that the implementations under test have already gone through functional tests.

Three standard classification techniques, namely Bayesian networks, *k*-nearest neighbor, and neural networks, are frequently used in classification problems. For instance, references [7, 8, 17, 19, 24] have reported their applications in various domains ranging from weather forecasting to data mining in medical databases.

Our thesis is to apply the classification techniques to identify faults in multimedia systems. There are, in general, two phases in the classification process, namely training and testing. In the training phase, information will be collected from a reference system, which is expected to be free from faults, and from different classes of systems with known faults. For each system, values of a set of attributes are collected by running the system several times with the same input. In our study, we collect (*a*) the difference in start times between two media objects, (*b*) the difference in end times between two media objects, (*c*) the duration of each of the two media objects, and (*d*) the maximum memory utilization. As discussed in the previous paragraphs, fuzziness in multimedia systems can be due to various reasons such as system load or faults in temporal relationships. The attribute "maximum memory utilization", besides placing the training and testing systems under different system loading, is a preliminary attempt to capture non-temporal attributes for identifying faults in multimedia systems. The list of attributes collected are by no means exhaustive. More discussions can be found in Section 5.

In the testing phase, values of the same set of attributes will be collected several times from the implementation under test (IUT). Each set of attribute values from the IUT is compared with those collected in the training phase. Based on the results, the probabilities that the IUT resembles the respective training systems will be computed. More details of the experiment will be described later.

3.1. X-Smiles and SMIL

The X-Smiles system [5], is used as the IUT in our experiments. SMIL documents are used as the inputs to X-Smiles. *X-Smiles* is an XML-browser developed in Java by the Telecommunications Software and Multimedia Laboratory at Helsinki University of Technology. It is an open source browser that can display documents written in various XML languages. SMIL is one of the XML languages supported by X-Smiles. X-Smiles version 0.71 is used in the experiments to test the temporal relationships among media objects.

The *Synchronized Multimedia Integration Language* (*SMIL*) [3] is designed by the World Wide Web Consortium (W3C) to provide a simple means of authoring interactive audiovisual presentations. It is targeted for integrating audios, videos, images, and text for "rich media" presentations. It is an XML language to describe the temporal relationships of such presentations, the layouts of the presentations, and associate hyperlinks with other media objects.

3.2. Experimental Setup

In order to conduct the experiments, the following software have been installed: (*i*) Java Development Kit 1.3, (*ii*) Java Media Framework 2.1.1, and (*iii*) X-Smiles 0.71.

As mentioned in Section 2, the three temporal relationships in Figure 3 are fundamental. Hence, we have developed three SMIL documents with faults related to them. Two objects are specified in each document. One is a slide show and another one is an audio narration. The slide show is in animated GIF format with dimensions of 640 $\times$ 480 pixels, while the audio narration is in WAV format with a sampling rate of 44.1 kHz. The three SMIL documents are

***Set*1 (Simultaneous):** Slide show *starts_after* audio narration immediately by 0 delay and slide show *ends_ahead_of* audio narration also by 0 delay;

***Set*2 (One During Another):** Audio narration *starts_after* slide show and audio narration *ends_ahead_of* slide show; and

***Set*3 (One Before Another):** Slide show *before* audio narration

as shown in Figure 5.

For *Set*1 and *Set*2, the selected features include the duration of the slide show, the duration of audio narration, the difference in start times between the slide show and its audio narration, and the difference in end times between the slide show and its audio narration. For *Set*3, the features include the duration of the slide show, the duration of audio narration, the start time of the slide show, and the difference between the end time of the slide show and the start time of the audio narration. The maximum memory utilization of the system is also recorded in all the sets.

The performances of the reference system used for the training include both the situations when the CPU is almost idle and when it is extremely busy. To simulate a low CPU utilization situation, all other utilities and applications are turned off. To simulate a high CPU utilization situation, a CPU-hungry utility is kept running to ensure that the system is heavily loaded.

The experiments have been carried out on different machines. For the training phase, it is conducted on a standalone Pentium Celeron 500 MHz computer with 256 MB of memory. For the testing phase, we have used two machines. One is the training machine and the other is a Pentium Celeron 550MHz computer with 512 MB of memory. All the machines are run under Windows 2003 Enterprise Edition.

The details of the four types of systems used in the training phase are as follows:

***Sys*0:** This is the reference system.

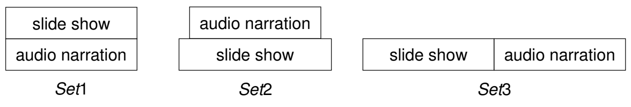

Figure 5. Test cases with different temporal relationships.

Next, in order to simulate the faulty systems, three different types of fault have been seeded into the reference system:

***Sys*1:** A priority error is simulated. It reduces the priority of execution threads. It simulates an inappropriate priority configuration in multimedia systems.

***Sys*2:** A pagination error is simulated. It causes X-Smiles to consume a large amount of system memory and, hence, the system will have excessive pagination. This fault can be the result of inappropriate resource handling in multimedia systems. Since the effects of system operations such as pagination are non-deterministic and may vary from system to system, it is difficult to replay the situations and detect this kind of failure.

***Sys*3:** A start time error is simulated. It delays the start time of any given media object by a fixed parameter. In this experiment, it is set to 10 seconds. It is one of the simplest errors found in multimedia systems. It models a multimedia system having some erroneous internal operations or calculations that delay the start time of its media objects.

480 data are used for training and 1760 data for testing. For the training phase, 20 repeated experiments have been carried out for each combination of high-low CPU utilizations, the 3 sets of SMIL specifications, and the 4 types of systems. We have re-run the same experiment 20 times to cater for common-cause variations. The same training set is used in all classification techniques to ensure a fair comparison.

Similarly, in the testing phase, we group multiple runs of the same experiments as data suites. Each data suite is compared with the data collected in the training phrase using the three classification methods to be discussed later. The probabilities of resemblance with the respective training systems are computed accordingly. For each test suite, the classification results are recorded. Table 1 shows the classification result of a sample test suite. This particular example shows that *Sys*1 system has the highest probability (0.8) and, hence, we classify the IUT as a system with priority error.

One important issue in classification is the overfitting problem. *Overfitting* [22] refers to the situation in which a complex function is learned almost to perfection during training, but the classification results remain poor because all test cases, except the training data themselves, are regarded as dissimilar. In order to avoid overfitting, test data with larger error sizes and test data collected from another machine are also examined in our testing phrase.

Probability of *Sys*0	0
Probability of *Sys*1	0.8
Probability of *Sys*2	0.1
Probability of *Sys*3	0.1

Table 1. The classification result of a sample test suite.

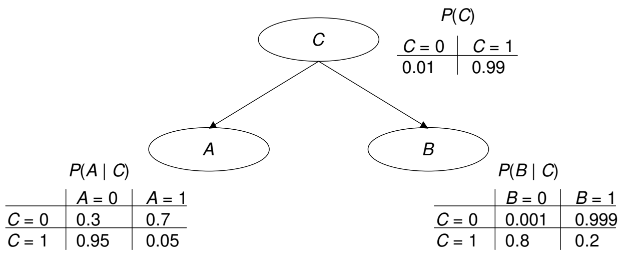

Figure 6. Example of Bayesian network.

3.3. Classification Techniques Investigated

Three common classification techniques have been used in this study.

3.3.1. Bayesian Networks

Bayesian networks (also known as *Bayesian belief networks*) [25, 26] is a probabilistic learning method to model uncertainties. A Bayesian network is a directed acyclic graph whose nodes represent the attributes of the training data. A Bayesian network can be represented by a triple (N, E, P), where N is a set of nodes, $E \subseteq N \times N$ is a set of edges, and P is a set of conditional probabilities [25, 26]. It will learn from the training data about the structure (namely, the interrelationships among attributes) and the set of conditional probabilities. For example, Figure 6 shows a simple Bayesian network that relates node C to nodes A and B separately. It also indicates that the probabilities of nodes A and B are affected by that of node C. The interrelationships and conditional probabilities are then used to classify the test data. Similar to other graphic languages like UML, a Bayesian network can allow users to present and reason about structural properties more easily than conventional mathematical or textual formulations.

In our experiments, Belief Network PowerPredictor [1]

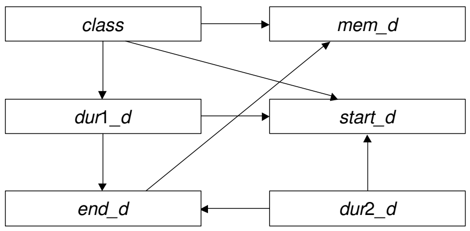

Figure 7. Example of generated Bayesian network structure.

is used. The training data are processed by a data preprocessor so as to convert them into a suitable format and to discretize any continuous attributes. Then, the training data (including structures and parameters) are passed to a belief network predictor to construct a belief network classifier from the data set, so that the network is to classify the test data. Figure 7 shows one of the belief networks generated by the constructor (according to *Set*1 under high CPU utilization). In this figure, *class* is the result of the classified type of faulty system; *mem* is the memory utilization; $dur1\_d$ is the duration of the slide show; $dur2\_d$ is the duration of the audio narration; $start\_d$ is the difference between the start times of the slide show and the audio narration; and end_d is the difference between the end times of the slide show and the audio narration. An arc between nodes in the graph represents a *direct dependency* (also known as a *casual relationship*) between the corresponding attributes. For example, there is a direct dependency between $dur1\_d$ and $start\_d$, but not between end_d and $start\_d$. Finally, we simply input the test data to obtain the test results according to the trained Bayesian network.

3.3.2. *k*-Nearest Neighbor

k-nearest neighbor was first introduced by Fix and Hodges [15] back in 1952. It is a non-parametric classification technique, which does not make any assumption about the underlying distribution of the data in question [14]. The algorithm is quite simple, because no training process is required. Nearly all the computation effort takes place at the testing phase.

Given n training data sets, let $\boldsymbol{x_i} = (a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(m)})$ represent the values of the attributes of the ith data set, where $i = 1, 2, \ldots, n$. A query with respect to a test case $\boldsymbol{x} = (a^{(1)}, a^{(2)}, \ldots, a^{(m)})$ can be input to the classifier, which will find a set $S' = \{\boldsymbol{x'_1}, \boldsymbol{x'_2}, \ldots, \boldsymbol{x'_k}\}$ of k training data from the training set $S = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$ such that

$$\max_{\boldsymbol{x'_i} \in S'} d(\boldsymbol{x'_i}, \boldsymbol{x}) \leq \min_{\boldsymbol{x_i} \in S \setminus S'} d(\boldsymbol{x_i}, \boldsymbol{x})$$

where $d()$ is the distance function.

In our experiment, we have implemented a k-nearest neighbor classifier in C++. It uses the standard Euclidean distance $d(\boldsymbol{x_i}, \boldsymbol{x}) = \sqrt{\sum_{j=1}^{m}(a_i^{(j)} - a^{(j)})^2}$ as the distance function. We have selected k to be 3. Since there is no training phase for this technique, we simply store the training data into the four classes *Sys*0, *Sys*1, *Sys*2, and *Sys*3. For every test data $\boldsymbol{x}$ in the testing phase, we compute the Euclidean distances of all the training data from $\boldsymbol{x}$. We then select 3 training data with the minimum distances from $\boldsymbol{x}$. If at least 2 of these 3 training data fall into one class, then the latter will be reported as the classification for $\boldsymbol{x}$. If there is no clear majority, then $\boldsymbol{x}$ is discarded.

3.3.3. Neural Networks

Neural networks [21] are designed to model the human brain, which consists of billions of neurons interconnected with one another. In neural networks, a *perceptron* models a neuron. It sums up a number of weighted inputs $(x_i)_{i=1,2,\ldots,n}$ to give $net = \Sigma_{i=0}^{n} w_i x_i$, and generates an output y via some transfer function, as shown in Figure 8. Usually, a sigmoid transfer function $1/(1+e^{-net})$ is used owing to its differentiable property. *Multilayer perceptrons* (*MLPs*) are a layered feed-forward network typically trained with static backpropagation as shown in Figure 9. Initially, the network chooses a set of random weights for each node. During the training phase, inputs are fed into the network and the actual outputs are compared with the expected results to estimate the errors. The network then adjusts the weights according to the feedback from the estimated errors. This process is known as *backpropagation*. With its remarkable ability to derive meanings from complex or imprecise data, neural networks can be used to extract patterns and detect trends that are too complicated to be noticed by humans or other computer techniques.

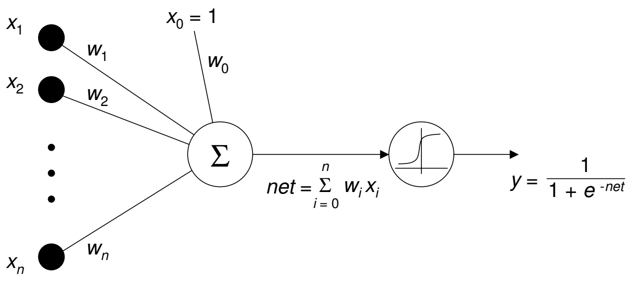

Figure 8. A perceptron (adapted from [21]).

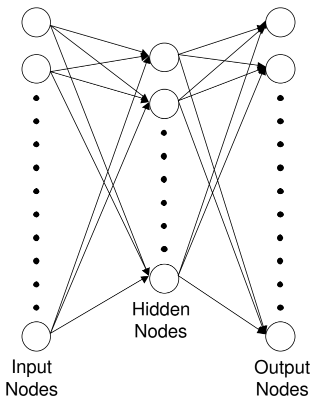

Figure 9. Multilayer perceptrons.

Classification Method	*Set* 1	*Set* 2	*Set* 3	Mean
Bayesian Networks	57.1%	75.0%	60.7%	64.3%
***k*-Nearest Neighbor**	60.7%	65.6%	67.9%	64.7%
Neural Networks	71.4%	40.6%	60.7%	57.6%

Table 2. Successful test results under low CPU utilization, by classes of test cases.

In our experiments, a neural network classifier NeuroSolutions [2] is used. It has been developed by NeuroDimension, a Computational Neural Engineering Lab at the University of Florida. MLPs are used by default as the model for training. In our experiment, there are five input nodes (representing the five attributes), four output nodes (representing the classes of systems), and one hidden layer (with four processing elements) in the neural network. First, inputs for the training data are recorded in a column-formatted ASCII file, while a separate file records the expected outputs. Then, we run the neural builder with the input data to train the neural network to perform the classification. During the testing phase, we simply input the test data to the neural network to obtain the test results.

4. Experimental Results

We have conducted experiments using 480 training data sets and 1760 test cases. The same training and test data have been used across all the investigated classification methods. As described before, three sets of test suites with different temporal relationships have been investigated. The experimental results are summarized in Tables 2 and 3. The numbers represent the percentage of test suites that are correctly identified by the respective classification methods.

In general, the results of the three classification techniques exhibit more or less the same percentage of accuracy in fault-identification. Under different CPU utilizations, the three classification techniques achieve an average accuracy of 69.3% in identifying the faults of the multimedia systems. Furthermore, we find that it is more likely to identify faults under high CPU utilization than under low utilization.

Classification Method	*Set* 1	*Set* 2	*Set* 3	Mean
Bayesian Networks	75.0%	75.0%	87.5%	79.2%
***k*-Nearest Neighbor**	78.6%	75.0%	75.0%	76.2%
Neural Networks	71.4%	75.0%	75.0%	73.8%

Table 3. Successful test results under high CPU utilization, by classes of test cases.

Under low CPU utilization, an average of 62.2% of the test suites can be correctly classified. We observe that for *Set*2, neural networks can only obtain an accuracy of 40.6%, which is far below the accuracy of the other two classification techniques. The reasons are still to be determined.

Under high CPU utilization, all the classification techniques can identify an average of 76.4% of the faults correctly. Table 4 shows the results according to the classes of faulty systems under high CPU utilization using each classification technique.

*Sys*3, which simulates a simple type of error, can be handled by any of the classification techniques with a success rate of 100%. As it is a simple fault, however, it may also be identified by conventional techniques such as tolerance zones. We can also see from Table 4 that the classification techniques only produce an average correctness of 54.8% for the faulty system *Sys*2. It may suggest that this kind of error is difficult to identify and requires more fine-tuned features. More investigations are needed on these aspects.

5. Discussions

By the nature of our experimental setup, if one conducts random guessing, the accuracy should be about 25% because of the four possible choices with a uniform distribution. Hence, the application of classification techniques to fault-identification is shown to be significant. On the other hand, the average success rate is only 69.3%. This indicates that there is still room for improvement:

(*a*) The most obvious enhancement is to train the classification systems using more data with a wider range of faults.

(*b*) Besides, the data collected may be in different units of measure. Normalization may be applied during data collection in the training phase. Informally, *normalization* is a process to fit the data in any range into a new range varying from 0 to 1. After normalization, all the data collected (no matter whether they are originally in units of "seconds" or "bytes") no longer have any unit of measure. The same normalization process should then be applied during data collection in the testing phase. Even when the same unit of measure is used, the collected data may fall into different scales. For example, "1" in a scale of 0 to 10 is similar in percentage to "10" in a scale of 0 to 100. However, they differ a lot when compared in absolute terms.

We have repeated our experiment by normalizing the data before performing classification. We find that the success rate is improved by an average of 7.6%. Hence, it is useful to apply normalization when the collected data are heterogeneous.

(*c*) In addition, the distribution of the attributes and the types of fault can obviously affect the accuracy of classification. We may check the data visually to "feel" the distribution. For instance, we may use a multidimensional scaling tool to determine whether they are geometrically distributed. *Multidimensional scaling* is a technique to produce geometric representations of data points showing their variations and dissimilarities [16]. Multidimensional scaling tools, such as XGvis provide graphic representations that help us gain an insight into the structure of the data points. Similar data points will be shown close to one another in the graph, while dissimilar data will not. *XGvis* [4] is free software that implements techniques for multidimensional scaling to graphically analyze the dissimilarities of the data [11]. When we find that the data are not sufficiently distinguishable despite multidimensional scaling, various remedial measures can be applied depending on the situations. For example, we may apply another classification technique, or gather more information from the training systems, or try to detect other types of fault.

As an illustration of (*c*), we may like to look into a comparison between low CPU and high CPU utilizations. Consider the attributes shown visually in the graphs of Figure 10. The two graphs, generated by XGvis, show the distribution of training data under low and high CPU utilizations, respectively. We observe that the selected attributes under high CPU utilization produces a more diversified geometrical distribution than under low CPU utilization. Hence, it will be easier for the classification techniques to identify faults under high CPU utilization (even though there are exceptions, such as the Bayesian network classifier case for *Set*2). It also shows the need to consider more attributes in situations with low CPU utilization.

Classification Method	*Sys*0	*Sys*1	*Sys*2	*Sys*3
Bayesian Networks	67.1%	82.9%	66.7%	100.0%
***k*-Nearest Neighbor**	72.6%	81.0%	51.2%	100.0%
Neural Networks	72.0%	76.8%	46.4%	100.0%

Table 4. Successful test results under high CPU utilization, by classes of systems.

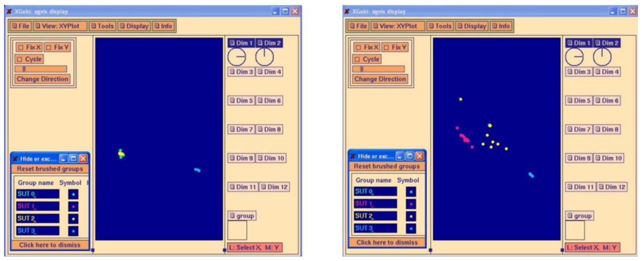

Figure 10. The left-hand figure shows the distribution of training data under low CPU utilization. The right-hand figure shows the distribution under high CPU utilization.

6. Conclusion

Testing can help us reveal failures in software systems. It will be even more useful if we can identify the type of fault when a failure occurs. It is not easy, however, to reveal the type of fault in multimedia systems because of non-deterministic and non-repeatable temporal relationships. We propose to apply classification techniques to tackle this problem. This paper reports on the results of the first step in this direction.

We have shown via empirical studies that classification techniques can learn non-deterministic characteristics from training data and identify the types of fault for multimedia systems. We consider temporal relationships and memory utilizations as the features for classification. We have studied empirically whether the effectiveness of identifying errors may be affected by the types of fault, the system loading, and the classification techniques. We find that some errors (such as *Sys*3) can be revealed easily while others (such as *Sys*2) are quite difficult to recognize. Based only on five attributes in our initial study, we have achieved an accuracy of 57.6 to 79.2% in revealing the types of fault in the presence of common cause variations. The empirical results are quite promising, although there is still room for improvement. For example, a normalization process may enhance the accuracy by an average of 7.6%.

As future work, we shall study the effectiveness of applying classification techniques to multimedia systems with more media objects and more temporal relationships. We shall examine the root causes of the interesting observations in the present study, with a view to selecting the more useful attributes of multimedia systems as features for classification, and improve on the classification process. We shall also investigate the feasibility of applying classification techniques to distributed multimedia systems. Finally, since multimedia systems has a lot of non-deterministic characteristics in common with other kinds of applications such as motion estimation systems, we shall also look into the extension of our technique to such systems.

Acknowledgments

The authors would like to Dr. W. K. Chan of The University of Hong Kong for his discussions and suggestions.

References

[1] Belief Network PowerPredictor. Available at http://www.cs.ualberta.ca/∼jcheng/bnpp.htm.

[2] NeuroSolutions. Available at http://nd.com.

[3] SMIL specification. Available at http://www.w3.org/AudioVideo.

[4] XGvis. Available at http://www.research.att.com/areas/stat/xgobi.

[5] X-Smiles. Available at http://www.xsmiles.org.

[6] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26: 832–843, 1983.

[7] L. Arbach, D. Lee, J. M. Reinhardt, and G. Fallouh. Mammographic masses classification: comparison between backpropagation neural network (BNN), k nearest neighbors (KNN), and human readers. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering* (*CCECE 2003*), volume 3, pages 1441–1444. IEEE Canada, 2003.

[8] S. Baek and K.-M. Sung. Fast k-nearest-neighbour search algorithm for nonparametric classification. *Electronics Letters*, 36 (21): 1821–1822, 2000.

[9] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 1990.

[10] G. Blakowski and R. Steinmetz. A media synchronization survey: reference model, specification, and case studies. *IEEE Journal on Selected Areas in Communications*, 14 (1): 5–35, 1996.

[11] A. Buja, D. F. Swayne, M. L. Littman, N. Dean, and H. Hofmann. XGvis: interactive data visualization with multidimensional scaling. To appear in *Journal of Computational and Graphical Statistics*. Also available at http://www.research.att.com/areas/stat/xgobi/papers/xgvis.pdf.

[12] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing without the need of oracles. *Information and Software Technology*, 45 (1): 1–9, 2003.

[13] S. C. Cheung, S. T. Chanson, and Z. Xu. Toward generic timing tests for distributed multimedia software systems. In *Proceedings of the 12th International Symposium on Software Reliability Engineering* (*ISSRE 2001*), pages 210–220. IEEE Computer Society Press, Los Alamitos, California, 2001.

[14] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13: 21–27, 1967.

[15] E. Fix and J. L. Hodges. Discriminatory analysis: nonparametric discrimination: small sample performance. Technical Report No. 11. Project No. 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas, 1952.

[16] P. J. F. Groenen. *The majorization approach to multidimensional scaling: some problems and extensions*. DSWO Press, Leiden University, Leiden, The Netherlands, 1993.

[17] J. Lee, R. C. Weger, S. K. Sengupta, and R. M. Welch. A neural network approach to cloud classification. *IEEE Transactions on Geoscience and Remote Sensing*, 28: 846–855, 1990.

[18] T. D. C. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communications*, 8 (3): 413–427, 1990.

[19] J. N. K. Liu and R. S. T. Lee. Rainfall forecasting from multiple point sources using neural networks. In *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics* (*SMC '99*), volume 3, pages 429–434. IEEE Computer Society Press, Los Alamitos, California, 1999.

[20] G. Lu. *Communication and computing for distributed multimedia systems*. Artech House, Boston, 1996.

[21] T. M. Mitchell. *Machine learning*. McGraw Hill, New York, 1997.

[22] F. Padberg, T. Ragg, and R. Schoknecht. Using machine learning for estimating the defect content after an inspection. *IEEE Transactions on Software Engineering*, 30 (1): 17–28, 2004.

[23] E. L. Russell, L. H. Chiang, and R. D. Braatz. *Data-driven techniques for fault detection and diagnosis in chemical process*. Springer, Berlin, 2000.

[24] J. L. Su, G. Z. Wu, and I. P. Chao. The approach of data mining methods for medical database. In *Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 4, pages 3824–3826. IEEE Press, New York, 2001.

[25] H. Ziv and D. J. Richardson. Bayesian-network confirmation of software testing uncertainties. Technical Report. University of California, Irvine, California, 1997.

[26] H. Ziv and D. J. Richardson. Constructing Bayesian-network models of software testing and maintenance uncertainties. In *Proceedings of the 13th IEEE International Conference on Software Maintenance* (*ICSM '97*), pages 100–109. IEEE Computer Society Press, Los Alamitos, California, 1997.