MESO: Supporting Online Decision Making in Autonomic Computing Systems

Eric P. Kasten, *Member*, *IEEE*, and Philip K. McKinley, *Member*, *IEEE*

Abstract—Autonomic computing systems must be able to detect and respond to errant behavior or changing conditions with little or no human intervention. Clearly, decision making is a critical issue in such systems, which must learn how and when to invoke corrective actions based on past experience. This paper describes the design, implementation, and evaluation of MESO, a pattern classifier designed to support online, incremental learning and decision making in autonomic systems. A novel feature of MESO is its use of small agglomerative clusters, called *sensitivity spheres*, that aggregate similar training samples. Sensitivity spheres are partitioned into sets during the construction of a memory-efficient hierarchical data structure. This structure facilitates data compression, which is important to many autonomic systems. Results are presented demonstrating that MESO achieves high accuracy while enabling rapid incremental training and classification. A case study is described in which MESO enables a mobile computing application to learn, by imitation, user preferences for balancing wireless network packet loss and bandwidth consumption. Once trained, the application can autonomously adjust error control parameters as needed while the user roams about a wireless cell.

Index Terms—Autonomic computing, adaptive software, pattern classification, decision making, imitative learning, machine learning, mobile computing, perceptual memory, reinforcement learning.

1 Introduction

Increasingly, software needs to adapt to dynamic external conditions involving hardware components, network connections, and changes in the surrounding physical environment [1], [2], [3]. For example, to meet the needs of mobile users, software in handheld, portable, and wearable devices must balance several conflicting and possibly crosscutting concerns, including quality-of-service, security, energy consumption, and user preferences. Applications that monitor the environment using sensors must interpret the knowledge gleaned from those observations such that current and future requirements can be met. *Autonomic computing* [4] refers to systems capable of addressing such situations through self-management and self-healing, with only high-level human guidance.

In recent years, numerous advances have been made in software *mechanisms* to support dynamic adaptation and autonomic computing; a recent survey can be found in [1]. However, new approaches to *decision making* are also needed to enable software to capture the relative importance of different inputs when confronting a dynamic physical world. For systems to learn from past experience and remember effective responses to the sensed environment, they must be able to filter an enormous number of inputs that may affect the decision. Moreover, many systems must make decisions in real time to prevent damage or loss of service. We argue that *perceptual memory*, a type of long-term memory for remembering external stimulus patterns [5], may offer a useful model for an important component of decision making in context-aware, adaptive software. The ability to remember complex, high-dimensional patterns that occur as a product of interaction between application users and the environment, and to quickly recall associated actions, can support timely, autonomous system response and even discovery of new or improved algorithms [6].

This paper presents MESO,[1] a perceptual memory system designed to support online, incremental learning, and decision making in autonomic systems. A novel feature of MESO is its use of small agglomerative clusters, called *sensitivity spheres*, that aggregate similar training patterns. Sensitivity spheres are partitioned into sets during the construction of a memory-efficient hierarchical data structure. This structure enables the implementation of a content-addressable perceptual memory system: instead of indexing by an integer value, the memory system is presented with a pattern similar to the one to retrieve from storage. Moreover, the use of sensitivity spheres facilitates a high rate of data compression, which enables MESO to execute effectively in resource-constrained environments. Additional benefits of MESO include: incremental training, fast reorganization, high accuracy, and lack of dependence on a priori knowledge of adaptive actions. Each of these benefits is important to online decision making.

After describing the design and operation of MESO, we demonstrate its accuracy and performance by evaluating it strictly as a pattern classifier. In these experiments, cross-validation experiments are used to determine accuracy using standard data sets. The performance of MESO, in terms of accuracy and execution time, compares favorably to that of other classifiers across a wide variety of data sets.

• *The authors are with the Department of Computer Science and Engineering, Michigan State University, 3115 Engineering Building, East Lansing, MI 48824. E-mail: {kasten, mckinley}@cse.msu.edu.*

Manuscript received 7 Oct. 2005; revised 23 June 2006; accepted 25 Sept. 2006; published online 19 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0467-1005.
Digital Object Identifier no. 10.1109/TKDE.2007.1000.

1. The term MESO refers to the tree algorithm used by the system (Multi-Element Self-Organizing tree).

Next, we describe how MESO enables software decision making in an audio streaming application that can *imitatively learn* [7], [8], [9] how to adapt to changing network conditions, such as loss rate, packet delay, and bandwidth availability. This application, called XNetApp, learns how to adapt through interaction with a user. Specifically, we trained the XNetApp how to respond to dynamic error conditions on a wireless network, and then tested its decision making ability by letting it execute autonomously. This proof-of-concept study demonstrates that perceptual memory systems, such as MESO, can play an effective role in the software decision-making process.

In [9], we described results of a preliminary study of MESO. This paper expands on that report in several ways. First, we describe related work on the application of machine learning to software decision making. Second, we provide a more comprehensive presentation of constituent algorithms and data structures used in MESO, as well as experimental results that help to elucidate issues related to the size and growth of sensitivity spheres. Third, all baseline and comparative experimental results presented in the paper were produced using a new version of MESO, written in C++ instead of Java. Fourth, the experimental results are expanded to included training and testing times and a comparison with a sequential-search version of MESO. Fifth, in addition to the seven data sets used in [9], we also evaluate MESO on the MNIST data set. Sixth, in addition to comparing MESO to three flavors of the IND [10] classifier, we also compare MESO directly against HDR [11], a classifier that uses incremental training. The results show that MESO can be trained and tested significantly faster than HDR. Moreover, MESO accuracy surpasses that of incremental HDR while comparing favorably with batch-trained HDR.

The remainder of this paper is organized as follows: Section 2 discusses background and related work. Section 3 describes MESO's clustering algorithm and the role of sensitivity spheres; three data compression methods that leverage MESO's internal structure are also introduced. Section 4 presents experimental results that assess MESO performance (accuracy, compression rate, and execution time) on eight standard data sets. MESO performance is also compared directly with that of other classifiers. Section 5 describes the mobile computing case study using XNetApp. Finally, Section 6 presents our conclusions and discusses future directions.

2 Background and Related Work

In this work, we explore clustering and pattern classification methods for associating adaptive responses with observed or sensed data. The embodiment of this approach is a clustering algorithm [12], [13] that produces a model of environmental stimuli. As shown in Fig. 1, two basic functions compose the operation of MESO: *training* and *testing*. During training, patterns are stored in perceptual memory, enabling the construction of an internal model of the training data. Each training sample is a pair (x_i, y_i), where x_i is a vector of continuous, binary, or nominal values, and y_i is an application specific data structure containing metainformation associated with each

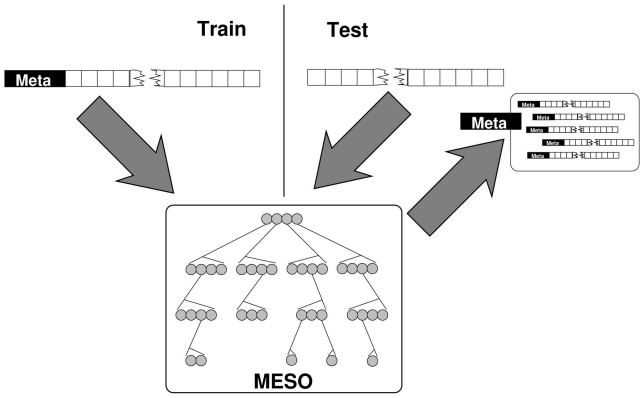


Fig. 1. High-level view of MESO.

pattern. Metainformation can be any data that is important to a decision-making task, such as the codification of an adaptive action to be taken in response to certain environmental stimuli. MESO can be used strictly as a pattern classifier [12] if an a priori categorization is known during training. In this case, the metainformation need only comprise of a label assigning each pattern to a specific real-world category. However, where many classifiers leverage categorical labels to better classify training samples, MESO does not rely on labels or any other type of metainformation, but instead incrementally clusters the training patterns in a label independent fashion.

Like many clustering and classifier designs, MESO organizes training patterns in a hierarchical data structure for efficient retrieval. Once MESO has been trained, the system can be queried using a pattern without metainformation. MESO tests the new pattern and returns either the metainformation associated with the most similar training pattern or a set of similar training patterns and their metainformation. In some domains, it may not be possible to collect a representative set of training samples a priori, so *incremental learning* is required. This process uses an estimation function f_i, which is a function of the first i samples, and which is constructed incrementally using the previous estimator f_{i-1} and the current pattern (x_i, y_i).

Research in clustering and pattern classification is a very active field of study [14], [15], [16], [17]. Recently, a number of projects have addressed clustering and classification of large data sets, a characteristic of decision making for autonomic software. Tantrum et al. [18] consider model-based refractionation for clustering large data sets. Yu et al. [19] use an hierarchical approach to clustering using support vector machines (SVMs). Kalton et al. [20] address the growing need for clustering by constructing a framework that supports many clustering algorithms. Methods for online clustering and classification have also been explored [21], [22], [23]. Like MESO, methods that address large data sets and online learning may provide a basis for a perceptual memory system. However, to our knowledge, MESO is the first to consider the combined tradeoffs of data intensity, time sensitivity, and accuracy with respect to memory systems within a decision-making environment.

Some of the concepts used in MESO are reminiscent of other clustering systems and, in some cases, a complementary relationship exists. For example, like MESO, M-tree [24] partitions data objects (patterns) based on relative distance. However, MESO uses an incremental heuristic to grow sensitivity spheres rather than splitting fixed sized nodes during tree construction. Moreover, rather than select database routing objects for directing the organization of the tree, MESO introduces the concept of pivot spheres for this purpose. BIRCH [25] also uses hierarchical clustering while iteratively constructing an optimal representation under current memory constraints. Where BIRCH mainly addresses data clustering when memory is limited, MESO attempts to balance accuracy, compression, and training and testing times to support online decision making. MESO may benefit from BIRCH's concept of *clustering features* as an efficient representation of training patterns, while BIRCH may benefit from MESO's approach to growing sensitivity spheres. Data Bubbles [26] focuses on producing a compressed data set representation while avoiding different types of cluster distortion. Its data analysis and representation techniques might enable alternative approaches to representing and compressing sensitivity sphere data in MESO, whereas MESO's growth and organization of sensitivity spheres could provide an efficient data structure for application of these techniques.

Other works have explored the use of statistical methods and pattern classification and clustering techniques in learning systems, including those that enable a system to learn online through interaction with the physical world. For example, Hwang and Weng [11] developed hierarchical discriminant regression (HDR) and applied it successfully as part of the developmental learning process in humanoid robots. Notably, HDR provides an hierarchical discrimination of features that helps limit the impact of high-dimensional feature vectors, enhancing the ability of the system to correctly classify patterns. However, as will be shown in Section 4, HDR requires significantly more time for training and testing than does MESO. In addition, Ivanov and Blumberg [27] developed the layered brain architecture, which was used for the construction of synthetic creatures, such as a "digital dog." That project used clustering and classification methods to construct perceptual models as part of the dog's developmental learning system. A notable aspect of the layered brain project is the use of compression to limit the effect of large training sets on memory consumption and processing power requirements. MESO also uses compression, but applies it to individual sensitivity spheres in order to maintain high accuracy in the face of data loss.

Our case study with MESO and XNetApp complements other studies of *imitative learning*, where a learner acquires skills by observing and remembering the behavior of a teacher. For example, Amit and Matarić [8] used hidden Markov models (HMMs) to enable humanoid robots to learn aerobic-style movements. The ability of the system to reconstruct motion sequences is encouraging, demonstrating the potential importance of imitative learning. Jebar and Pentland [7] conducted imitative learning experiments using a wearable computer system that included a camera and a microphone. A human subject was observed by the system during interactions with other people. The observed training data was used to train an HMM. Later, the system was allowed to respond autonomously when presented with visual and audio stimuli, demonstrating a limited ability to reproduce correct responses. However, since learning by observing real human behavior is very complex, even limited recognizable response is significant and promising. The development of MESO complements these approaches by providing a fast and memory-efficient means to classify internal state under external conditions.

```
initialize cluster centers, δ
input pattern x(t)
find nearest center, e.g., w_i
if d(x_i, w_i) ≤ δ
  update cluster center
else
  create new center w_j = x(t)
next pattern
end
```

Fig. 2. Leader-follower algorithm (adapted from Duda and Hart [12]).

Finally, researchers have applied data clustering and classification methods to other aspects of autonomic computing, such as fault detection and optimization of algorithms. Fox et al. [28] used data clustering to correlate system faults with failing software components. Once the failing components were identified, they could be selectively restarted, avoiding a complete system reboot while shortening mean time to recovery. Geurtz et al. [29] considered several machine learning algorithms for identifying if a system is running atop a wired or wireless network. This method enables the autonomous adaptation of the TCP protocol to address dynamic network conditions. It is anticipated that similar systems can use MESO for automated fault detection or optimization when the software is faced with the uncertainty found in dynamic environments.

3 MESO Design and Operation

If categorical labels are known during training, MESO can function as a pattern classifier that incrementally classifies environmental stimuli or other data while accommodating very large data sets. Prior to developing MESO, we conducted experiments using the HDR classifier [11] for this purpose. The insights gained from those experiments led to our design of MESO. MESO incrementally constructs a model of training data using a data clustering approach whereby small clusters of patterns, called *sensitivity spheres*, are grown incrementally. These sensitivity spheres are organized in an hierarchical data structure, enabling rapid training and testing, as well as significant data compression, while maintaining high accuracy. In this section, the details of MESO's core algorithm and data structures are discussed. MESO is based on the well-known leader-follower algorithm [30], an online, incremental technique for clustering a data set. The basic operation of the leader-follower algorithm is shown in Fig. 2. A training pattern

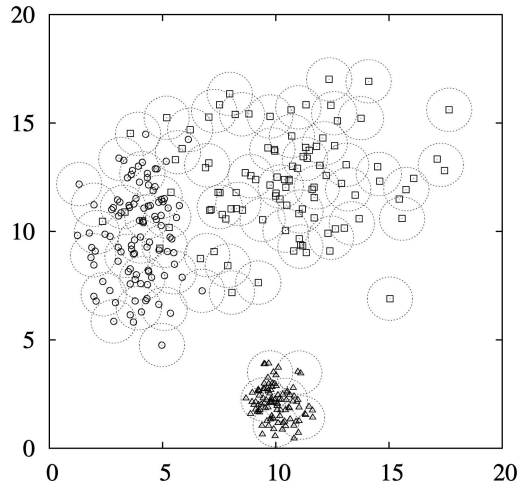


Fig. 3. Sensitivity spheres for three 2D Gaussian clusters. Circles represent the boundaries of the spheres as determined by the current δ. Each sphere contains one or more training patterns, and each training pattern is labeled as belonging to one of three categories (circle, square, or triangle).

within distance δ of an existing cluster center is assigned to that cluster; otherwise, a new cluster is created.

Traditionally, the value of δ is a constant initialized based on a user's understanding or experience with the data set at hand. However, this approach makes it difficult to generalize the leader-follower algorithm to arbitrary data sets. We address this issue in MESO by computing the value of δ incrementally and by organizing the resulting clusters using a novel hierarchical data structure, as described below.

3.1 Sensitivity Spheres

In adaptive software, training patterns comprise observations related to quality of service or environmental context, such as network bandwidth or physical location. The quantity of training patterns collected while a system executes may be very large, requiring more memory and processing resources as new patterns are added to the classifier. Unlike the traditional leader-follower algorithm, in MESO, the value of δ changes dynamically, defining the sensitivity spheres, which are small agglomerative clusters of similar training patterns. Effectively, the value of δ represents the sensitivity of the algorithm to the distance between training patterns. Fig. 3 shows an example of sensitivity spheres for a 2D data set which comprises three clusters. A sphere's center is calculated as the mean of all patterns that have been added to that sphere. The δ is a ceiling value for determining if a training pattern should be added to a sphere, or if creation of a new sphere is required. As defined by the δ value, sphere boundaries may overlap, however, each training pattern is assigned to only one sphere, whose center is closest to the pattern.

3.2 MESO Tree Structure

As with many classifiers, MESO uses a tree structure to organize training patterns for efficient retrieval. However, the MESO tree, depicted in Fig. 4, is novel in that its organization is based on sensitivity spheres. A MESO tree is built starting with a root node, which comprises the set of all sensitivity spheres. The root node is then split into subsets of similar spheres which produces child nodes. Each child node is further split into subsets until each child contains only one sphere. Many clustering algorithms construct a tree by agglomerating individual patterns into large clusters near the root of the tree, and then splitting these clusters at greater tree depths. Reorganizing such a tree requires processing of the training patterns directly. In contrast, MESO's consolidation of similar patterns into sensitivity spheres enables construction of a tree using only spheres, rather than individual patterns. Moreover, a MESO tree can be reorganized using only existing sensitivity spheres and, hence, more rapidly than approaches that require direct manipulation of patterns.

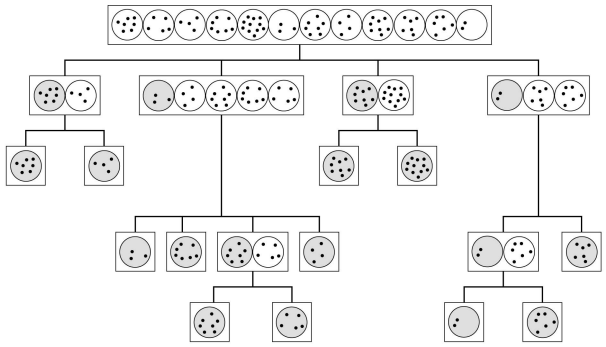


Fig. 4. MESO tree organization. The rectangles are partitions and the shaded spheres are partition pivots. Partitions are split successively until a leaf is formed where a partition contains only one sphere.

The set of sensitivity spheres for a data set is partitioned into subsets of similar spheres during the construction of a MESO tree. Each node of the tree contains one such subset, called a *partition*. Fig. 5 shows the algorithm for building a MESO tree from existing sensitivity spheres. The parameters for this algorithm include: q, the number of children per tree node; p, a partition pivot sphere; *parent*, the parent node for a set of children; *root*, the root node of the tree; and *part*, the partition associated with a *parent* node. The algorithm is recursive, starting at the *root* of the tree with a partition (*part*)

```
begin initialize q, p = nil, root, part
  splitpartition(q, p, root, part)

procedure splitpartition(q, p, parent, part)
  if part has a cardinality > 1
    select q pivots from part including
       p, p_1..p_q
    create q subpartions, part_1..part_q
    foreach s_i in part do
      find the nearest p_j pivot and add
        s_i to part_j
    done
    foreach p_j, part_j pair do
      create a child node
      add p_j to child
      add child to parent
      splitpartition(q, p_j, child, part_j)
    done
  endif
```

Fig. 5. Building a MESO tree from sensitivity spheres.

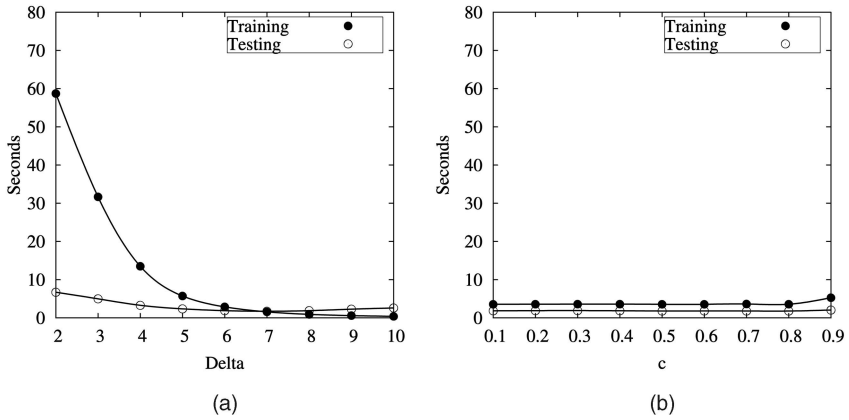


Fig. 6. Training and testing time for the letter data set (see Section 4.1). (a) Using fixed δ. (b) Using dynamic δ.

comprising all spheres in the tree. Each call to *splitpartition* divides *part* into q smaller partitions and assigns these partitions as children of the *parent* node. The processes terminates when a partition contains only one sphere. When a partition is divided, the first sphere in each of the q segments is identified as a *pivot*, which is used subsequently in assigning other spheres to that partition. Specifically, for a sphere to be added to a partition requires that the sphere be nearer to that partition's pivot than to the pivot of any other child node. Intuitively, this algorithm can be viewed as a q-way heap sort that organizes sensitivity spheres according to their similarity. The parameter q can be set to any integer value ≥ 2 and, in our experience, has limited impact on the accuracy of retrieving patterns from MESO during testing. In the experiments described in Sections 4 and 5, we set $q = 8$.

As a result of this process, each nonleaf node in a MESO tree has one or more children, each comprises a subset of the parent's sensitivity spheres. Smaller partitions provide finer discrimination and better classification of test patterns. Moreover, the partitioning of sensitivity spheres produces a hierarchical model of the training data. That is, each partition is an internal representation of a subset of the training data that is produced by collecting those spheres that are most similar to a pivot sphere. At deeper tree levels, parent partitions are split, producing smaller partitions of greater similarity.

To classify a test pattern, the pattern is compared with a pivot, starting at the root, and following one or more paths of greatest similarity. At a leaf node, a label is returned indicating the category to which the test pattern most likely belongs. The MESO tree can be constructed incrementally, enabling MESO to be trained and tested during simultaneous interaction with users or other system components.

3.3 Sensitivity Sphere Size

An important consideration in building an effective MESO tree is the appropriate value of δ to use in defining sensitivity spheres. Our experiments show that training and testing time are influenced by the choice of δ. For example, Fig. 6a shows results for the letter data set (discussed further in Section 4.1), with δ fixed at various values. If δ is too small, training time increases dramatically. If δ is too large, testing time increases (more evident for larger data sets). Moreover, data set compression requires a proper value of δ to balance the tradeoff between compression rate and accuracy.

To address this issue, the value of δ is adjusted incrementally as MESO is trained. The δ *growth function* balances sphere creation rate and sphere size. Fig. 7 shows the algorithm for construction of sensitivity spheres from training patterns. This algorithm begins by initializing the sensitivity δ, the first sensitivity sphere mean vector (u_1), and the first sensitivity sphere (s_1) to 0, x_1, and empty, respectively. Then, for each pattern (x_j), the closest sphere mean vector is located. If the distance between x_j and the nearest sphere mean is less than or equal to δ, then x_j is added to the sphere and the sphere mean recalculated. If the distance between the closest sphere mean and x_j is greater than δ, then the δ is grown, then a new sphere is created for x_j and an associated mean vector is initialized.

A good $grow_\delta$ function needs to balance sphere creation with sphere growth. Rapid growth early in the training process can produce few spheres with very large δs, creating a coarse-grained, inefficient representation. However, slow growth produces a large number of very small spheres, and the resulting tree is expensive to search. In the MESO implementation reported here, the δ growth function is:

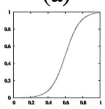
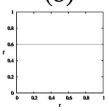
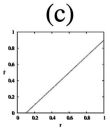
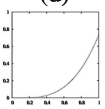
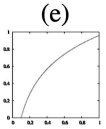
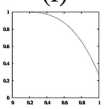
$$grow_\delta = \frac{(d-\delta)\frac{\delta}{d}f}{1+\ln(d-\delta+1)^2},$$

where d is the distance between the new pattern and the nearest sensitivity sphere. The $\frac{\delta}{d}$ factor scales the result

```
begin initialize δ = 0, u1 = x1, s1
  foreach xj sample do
    find the nearest ui for xj
    if distance from ui to xj <= δ
      add xj to si
      recompute ui using samples in si
    else
      let δ = growδ
      create new si+1
      add xj to si+1
      let ui+1 = xj
    endif
  done
```

Fig. 7. Sensitivity sphere creation algorithm.

TABLE 1
Comparison of Six Different Activation Functions Using $c = 0.6$ for the Letter Data Set (See Section 4.1)

Data set	(a)  $\frac{1}{2} + \frac{\tanh(\frac{3r}{c} - 3)}{2}$	(b)  $c = 0.6$	(c)  $\frac{r}{c}$	(d)  $(\frac{r}{c})^3$	(e)  $log_{10}(\frac{9r}{c})$	(f)  $1 - (\frac{r}{c})^3$
Letter						
Accuracy%	90.6±0.3%	88.1±0.2%	87.9±0.2%	88.7±0.3%	87.9±0.2%	88.7±0.2%
Training (s)	1.8±0.0	2.0±0.0	1.8±0.0	2.2±0.0	1.7±0.0	2.5±0.0
Testing (s)	0.2±0.0	0.2±0.0	0.2±0.0	0.3±0.0	0.2±0.0	0.3±0.0
Spheres	570±22	659±2	563±3	803±8	510±3	953±4
MNIST						
Accuracy%	94.3±0.1%	94.8±0.1%	94.6±0.1%	95.0±0.1%	94.5±0.1%	95.2±0.1%
Training (s)	70.9±1.1	109.9±1.6	92.1±3.2	125.9±1.4	86.7±1.2	175.0±1.6
Testing (s)	6.7±0.2	9.3±0.3	8.6±0.7	10.2±0.1	7.9±0.2	11.3±0.3
Spheres	6279±11	9696±6	7050±8	10300±19	6300±2	13758±2

relative to the difference between the current δ and d. Intuitively, the denominator of $grow_\delta$ limits the growth rate based on how far the current δ is from d. If d is close to δ, then δ will grow to be nearly equal to d. However, if d is much larger than δ, then the increase will be only a small fraction of $d - \delta$. As such, δ growth is discouraged in the face of outliers, new experience, and widely dispersed patterns. Hence, when a new training pattern is distant from existing spheres, a new sphere is likely to be created for it.

The activation function, f, needs to balance the creation of new spheres with sphere growth. Table 1 depicts six candidate activation functions, where $r = \frac{spheres}{patterns}$ and c is a configuration parameter in the range [0, 1.0]. Increasing c moves the center of the activation function to the right. The statistics shown were generated using cross-validation (discussed further in Section 4.1) in conjunction with the letter and MNIST data sets. As shown, the functions in Tables 1b, 1d, and 1f produce a significantly larger number of sensitivity spheres than the other functions. However, a large sphere count inhibits compression (discussed further in Section 3.4) and exhibits higher training and testing times. The functions in Tables 1c and 1e produce fewer spheres, but exhibit somewhat lower accuracies or longer training and testing times than the function in Table 1a. Overall, the function in Table 1a shows the best balance between accuracy and training and testing times while producing a sufficiently small number of spheres to enable high compression. Intuitively, the function in Table 1a inhibits sensitivity sphere growth when the number of spheres is small compared to the number of patterns, but encourages rapid sphere growth when the number of spheres is large. The remaining experiments presented in this paper use the activation function in Table 1a, with parameter c set to 0.6.

Fig. 6b plots the measured training and testing time for the letter data set against the configuration parameter, c. The $grow_\delta$ function balances sphere production with sphere growth, producing good spheres for a wide range of values for c. Only for very large values of c is growth inhibited sufficiently to significantly impact training time. The $grow_\delta$ function promotes the production of trees that are comparable with good choices for fixed δ values.

3.4 Compression

Online learning is a data intensive process, and adaptive systems often must continue to function for long periods of time while responding to the sensed environment. The enormous amount of input data consumes substantial processing and storage resources, potentially inhibiting timely responses or impacting application performance. MESO uses lossy compression to limit the consumption of memory and processor cycles. Compression is applied on a per sensitivity sphere basis. That is, rather than trying to compress the entire data set using a global criterion, the patterns in each sensitivity sphere are compressed independent of other spheres. Since information about each sphere is retained, the effect of information loss on classifier accuracy is minimized. We implemented three types of compression, the evaluation of which is discussed in Section 4.2.

Means compression reduces the set of patterns in each sensitivity sphere to the mean pattern vector for each label. This is the most aggressive and simple of the compression methods. Moreover, the computational requirements are quite low.

Spherical compression is a type of boundary compression [27] that treats patterns on the boundaries between spheres as most important to the classification of test patterns. For each sphere, the feature values are converted to spherical coordinates. Along a given vector from the sphere center, only those patterns farthest from the sphere center are kept.

Orthogonal compression removes all the patterns that are not used for constructing an orthogonal representation of a sphere's patterns. The idea is to keep only those patterns that are most important as determined by their orthogonality. Patterns that represent parallel vectors in m-dimensional space are removed.

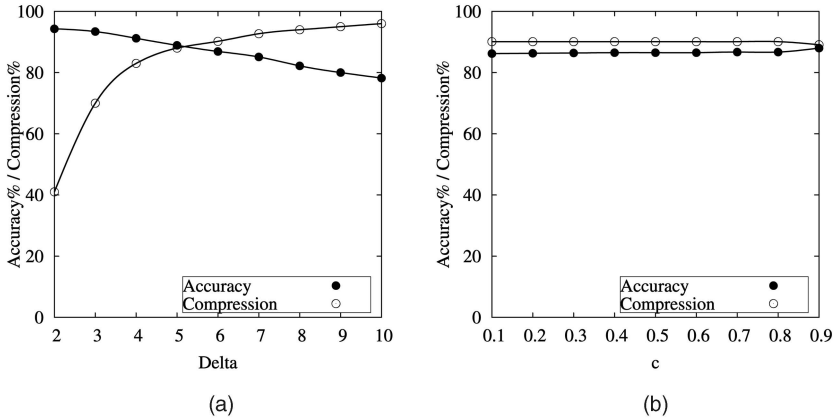


Fig. 8. Effect of means compression on training and testing times for the letter data set, using fixed and variable δ. (a) Accuracy and compression, fixed δ. (b) Accuracy and compression, variable δ.

Using compression requires some consideration of δ growth. As shown in Fig. 8a, accuracy decreases with higher compression rates. Moreover, the compression rate is directly influenced by the value of δ. That is, if the sensitivity sphere δ is very large and few spheres are produced, compression is high and too much information will be lost during compression. However, if the δ is very small, very little compression is possible.

To avoid growing overly large spheres in the face of compression, we modified the activation function f to be:

$$f = \frac{1}{2} + \frac{\tanh(\frac{3r}{\max(v,c)} - 3)}{2},$$

where v is the compression rate, defined as the fraction of patterns removed during compression. Under high compression rates, using v instead of c as the center point of the activation function causes the Sigmoid curve to move to the right, further inhibiting sphere growth. Fig. 8b plots the accuracy and compression rate for experiments on the letter data using means compression and the modified activation function. Accuracy and compression rate remain high for a wide range of c values. Only very large values of c cause a drop in compression rate, along with a slight increase in accuracy.

3.5 Complexity

Table 2 shows the space and time complexities for training MESO and several well-known clustering algorithms [31]. In this table, n is the number of patterns, k is the number of clusters, and l is the number of iterations to convergence.

TABLE 2
Space and Time Complexities for MESO and Several Other Clustering Algorithms [31]

Algorithm	Time	Space
MESO	$O(n \log_q k)$	$O(n)$
leader	$O(kn)$	$O(k)$
k-means	$O(nkl)$	$O(k)$
ISODATA	$O(nkl)$	$O(k)$
shortest spanning path	$O(n^2)$	$O(n)$
single-link	$O(n^2 \log n)$	$O(n^2)$
complete-link	$O(n^2 \log n)$	$O(n^2)$

Without compression, MESO has a worst-case space complexity of $O(n)$, comparable to the shortest spanning path algorithm. MESO's memory consumption can be significantly reduced with compression, as shown in the next section.

Intuitively, time complexity for training can be considered in terms of locating the sensitivity sphere nearest to a new pattern and adding the pattern to that sphere. If a sufficiently close sphere cannot be found, a new sphere is created. Locating the nearest sphere is an $O(\log_q k)$ operation. This search must be completed once for each of n patterns. Each pattern must also be added to a sensitivity sphere, and k sensitivity spheres must be created and added to the MESO tree. Assuming an appropriate value of δ and a data set of significant size, this process yields a complexity of $O(n \log_q k) + O(n) + O(k) + O(k \log_q k)$ which reduces to $O(n \log_q k)$.

The search complexity for classifying a test pattern using MESO is $O(\log_q k) + O(\bar{s})$ for a balanced tree, where q is the maximum number of children per node, $\bar{s}$ is the average number patterns agglomerated by a sensitivity sphere, and k represents the number of sensitivity spheres produced. The $\bar{s}$ component represents the number of operations required to assign a category label once the most similar sensitivity sphere has been located. Thus, the worst-case search complexity occurs when only one cluster is formed and the search algorithm degenerates into a linear search of $O(n)$. Conversely, a best-case search complexity of $O(\log_q n)$ occurs when one sensitivity sphere is formed for each training pattern.

4 MESO Assessment

In this section, we evaluate MESO as a pattern classifier on several standard data sets in cross-validation experiments. First, we describe the data sets used in the experiments and the experimental procedures. Next, we present baseline results that evaluate the accuracy of MESO, the training and testing time needed, and the effects of the three compression methods described earlier. Finally, to benchmark performance, we compare MESO performance to that of other classifiers, specifically, three versions of IND [10],

TABLE 3
Data Set Characteristics

Data Set	Patterns	Features	Labels
Iris	150	4	3
ATT Faces	360	10,304	40
Multiple Feature	2,000	649	10
Mushroom	8,124	22	2
Japanese Vowel	9,859	12	9
Letter	20,000	16	26
MNIST	70,000	784	10
Cover Type	581,012	54	7

which uses batch training, and HDR [11], which can be configured to use either batch or incremental training.

4.1 Data Sets and Experimental Method

Table 3 lists the eight data sets used to assess MESO. The number of patterns and features per pattern are shown for each data set, along with the number of distinct labels (or classes) of patterns. Six of the data sets were retrieved from the UCI [32] and KDD [33] machine learning repositories. The exceptions are AT&T faces [34], acquired from AT&T Laboratories Cambridge, and MNIST [35], downloaded from http://yann.lecun.com/exdb/mnist/.

These sets represent a wide variety of data types and characteristics. The iris data set [36] comprises just 150 patterns from three classes, each class representing a type of iris plant. The classification task is to correctly identify the type of iris by the length and width of the flower's sepals and petals. The AT&T faces data set [34] is also relatively small, and comprises 360 images of 40 different human subjects. However, the number of features (each of 10,304 image pixels) is very large. The classification task is to identify the subject of the image from the pixel values.

Three data sets involve numbers and letters. Patterns in the multiple feature data set [37], [38] consist of features that describe 10 handwritten numerals extracted from Dutch utility maps. Examples include morphological features, Fourier coefficients, and pixel averages. The classification task is to identify a digit from these features. The MNIST data set [35] also comprises features of handwritten digits, and the task is to identify the digit. However, the features are the 784 integer pixel values, and the number of patterns is much larger than in the multiple feature data set. The letter data set [39] contains 20,000 patterns, each comprises 16 integer measurements of features such as width, height, or mean pixel values. The classification task is to classify each pattern as one of the 26 letters in the Latin alphabet.

The mushroom [40] and Japanese vowel [41] data sets are similar in size and feature count, but very different in content. Each pattern in the mushroom data set comprises 22 nominal values (alphabetic characters) that represent mushroom features such as cap shape or gill attachment. Since MESO does not address nonnumeric attributes explicitly, each alphabetic character is converted to its numeric ASCII value. The binary label associated with a pattern indicates whether the mushroom is poisonous or edible. The Japanese vowel data set comprises 270 time series blocks, where each block consists of a set of records. Each record contains 12 continuous measurements of utterances from nine male speakers. The 9,859 patterns are produced by treating each record as an independent pattern and randomizing the data set. As such, no understanding of utterance order is retained. The classification task is to identify the speaker of each utterance independent of its position in a time series.

Finally, the cover type data set [42] comprises 581,012 patterns for determining forest cover type. Each pattern has 54 values, including: 10 continuous values, indicating features such as elevation and slope; four binary wilderness areas; and 40 binary soil types. The classification task is to identify which of seven forest cover types (such as spruce/fir or aspen) corresponds to a test pattern.

We tested MESO using cross-validation experiments as described by Murthy et al. [14]. Each experiment is conducted as follows:

1. Randomly divide the training data into k equal-sized partitions.
2. For each partition, train MESO using all the data outside of the selected partition. Test MESO using the data in the selected partition.
3. Calculate the classification accuracy by dividing the sum of all correct classifications by the total number of patterns tested.
4. Repeat the preceding steps n times, and calculate the mean and standard deviation for the n iterations.

In our tests, we set both k and n equal to 10. Thus, for each mean and standard deviation calculated, MESO is trained and tested 100 times.

4.2 Baseline Experiments

Table 4 presents results of cross-validation experiments using MESO to classify patterns in the eight data sets. Means and standard deviations are provided. Before discussing the results, let us briefly comment on the distance metric used. Since the use of sensitivity spheres effectively divides the larger classification problem into a set of smaller tasks, it turns out that a relatively simple distance metric, such as Euclidean distance, can be used to achieve high accuracy. Although we experimented with more complicated distance metrics (e.g., Mahalanobis), none achieved higher accuracy than Euclidean distance, which also exhibited shorter times for training and testing. Therefore, all experiments described here and in later sections use Euclidean distance.

Let us focus first on the results for experiments that do not use compression. MESO exhibits an accuracy of more than 90 percent on all the data sets, using either sequential or tree-based search. MESO's accuracy on the AT&T Faces and MNIST data sets, which contain high-dimensional, image data, indicates that MESO may be effective in computer vision applications. Compared to a sequential search of sensitivity spheres, use of the MESO tree structure reduces training and testing times in most cases. The improvement is particularly notable for large data sets. For MNIST, training time is improved by a factor of 18 and testing time by a factor of 20. For Cover Type, training time is improved by a factor of 18 and testing

TABLE 4
MESO Baseline Results Comparing a Sequential Search to MESO Tree Search, with and without Compression

Data set	Uncompressed		Compressed (Tree)		
	(Sequential)	(Tree)	Means	Spherical	Orthogonal
Iris					
Accuracy%	95.5±0.0%	96.1±1.4%	95.8±0.8%	95.1±1.3%	95.9±2.1%
Compression%	0.0%	0.0%	1.86±0.2%	0.0±0.0%	1.9±0.0%
Training (s)	0.0±0.0	0.03±0.0	0.03±0.0	0.03±0.0	0.03±0.0
Testing (s)	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
ATT Faces					
Accuracy%	97.3±0.0%	94.0±1.4%	93.5±1.6%	94.5±1.2%	93.7±1.4%
Compression%	0.0%	0.0%	0.0±0.0%	0.0±0.0%	0.0±0.0%
Training (s)	1.85±0.0	1.87±0.0	1.90±0.0	1.86±0.0	2.04±0.0
Testing (s)	0.39±0.0	0.08±0.0	0.08±0.0	0.08±0.0	0.08±0.0
Mult. Feature					
Accuracy%	95.0±0.0%	94.1±0.5%	94.2±0.4%	94.1±0.5%	94.4±0.5%
Compression%	0.0%	0.0%	0.3±0.0%	0.0±0.0%	0.3±0.0%
Training (s)	4.58±0.0	1.69±0.0	1.73±0.0	1.81±0.0	1.78±0.0
Testing (s)	0.99±0.0	0.07±0.0	0.07±0.0	0.07±0.0	0.07±0.0
Mushroom					
Accuracy%	100.0±0.0%	100.0±0.0%	100.0±0.0%	99.8±0.0%	99.9±0.0%
Compression%	0.0%	0.0%	90.2±0.0%	73.9±0.3%	90.2±0.0%
Training (s)	1.24±0.1	0.62±0.0	0.67±0.0	0.81±0.0	0.77±0.0
Testing (s)	0.14±0.0	0.05±0.0	0.05±0.0	0.05±0.0	0.05±0.0
Japanese Vowel					
Accuracy%	93.1±0.2%	91.5±0.3%	81.3±0.4%	90.2±0.3%	81.3±0.2%
Compression%	0.0%	0.0%	93.7±0.0%	28.3±0.2%	93.8±0.0%
Training (s)	0.30±0.0	0.39±0.1	0.41±0.0	0.89±0.0	0.49±0.0
Testing (s)	0.04±0.0	0.05±0.0	0.03±0.0	0.05±0.0	0.03±0.0
Letter					
Accuracy%	93.1±0.2%	90.6±0.3%	87.8±0.3%	90.1±0.2%	87.8±0.3%
Compression%	0.0%	0.0%	88.6±0.2%	23.6±0.2%	88.3±0.2%
Training (s)	1.83±0.2	1.27±0.0	1.42±0.0	2.28±0.0	1.77±0.0
Testing (s)	0.21±0.0	0.16±0.0	0.12±0.0	0.17±0.0	0.12±0.0
MNIST					
Accuracy%	96.5±0.0%	94.3±0.1%	93.3±0.1%	94.3±0.1%	93.3±0.1%
Compression%	0.0%	0.0%	86.5±0.0%	0.0±0.0%	86.5±0.0%
Training (s)	1307.22±9.7	70.94±1.1	73.35±1.3	179.53±5.4	78.65±1.4
Testing (s)	157.32±1.3	6.73±0.2	6.29±0.2	6.81±0.2	6.30±0.2
Cover Type					
Accuracy%	96.3±0.0%	96.1±0.0%	81.6±0.1%	95.2±0.0%	81.6±0.0%
Compression%	0.0%	0.0%	98.5±0.0%	50.2±0.0%	98.5±0.0%
Training (s)	1974.76±11.8	109.97±0.4	114.54±0.7	232.64±2.5	127.97±0.9
Testing (s)	227.08±1.4	18.74±0.1	11.14±0.1	17.29±.03	11.56±0.1

All tests were started with $\delta = 0.0$ and $c = 0.66$ and executed on a 2GHz Intel Xenon processor with 1.5 GB RAM running Linux. All experiments were conducted using cross-validation.

time by a factor of 12. Although using the hierarchical tree structure reduces the accuracy in most cases, typically between 0 percent to 4 percent, this tradeoff may be considered acceptable for applications where decision making is time sensitive.

Next, let us consider the results for experiments using data compression. The three methods (means, spherical, and orthogonal) had only minimal effect on the three smallest data sets, where sphere growth is inhibited early in the training process, producing spheres with few samples. However, the memory usage for these data sets is low. On the other hand, both the means and orthogonal methods were very effective in reducing the memory requirements for the five larger data sets (at least an 85 percent reduction in all cases), while retaining high accuracy. We attribute this behavior to the application of compression to individual sensitivity spheres, enabling the capture of the n-dimensional structure of the training data while limiting information loss. Spherical compression was the least effective in reducing memory usage; the translation of training patterns from Euclidean to spherical coordinates also adds to the cost of training.

Fig. 9 shows how MESO's accuracy and training times scale with the size of the training data set. To create these plots, each data set was first randomized and then divided into 75 percent training and 25 percent testing data. The training data was further divided into 100 segments. MESO was trained and then tested 100 times. During the first iteration, only the first segment was used for training; at each subsequent iteration, an additional segment was added to the training set. This process was repeated 10 times for each data set and the mean values calculated. The mean values are plotted in Fig. 9. As shown, MESO's accuracy increases rapidly during early training, and then slows but continues to improve as training continues. Training time increases linearly with respect to the size of the training data set.

4.3 Comparison with Other Classifiers

In this section, we compare MESO performance with that of the IND [10] and HDR [11], [43] classifiers. We note that

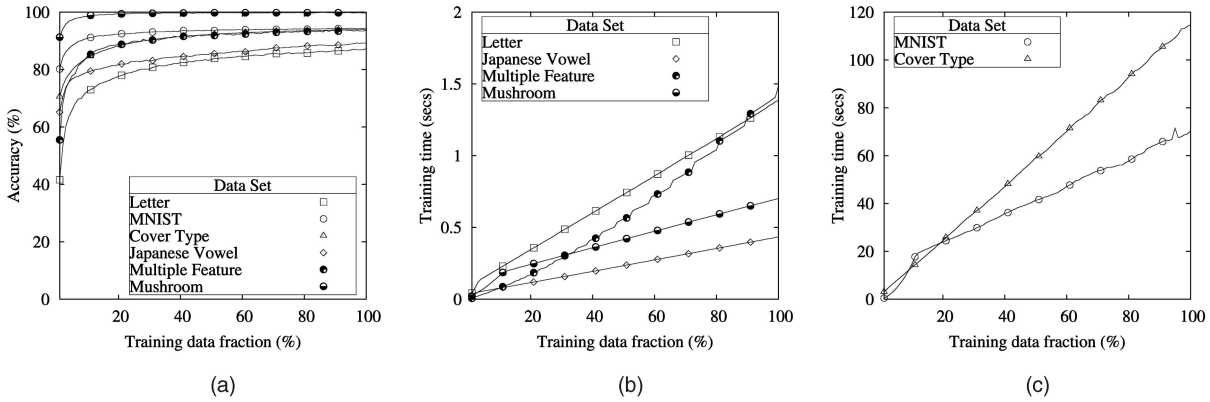


Fig. 9. Scalability with respect to training set size. For all data sets, typical standard deviations are less than 10 percent with respect to the corresponding mean accuracies and training times. (a) Accuracy. (b) Training time (small sets). (c) Training time (large sets).

MESO is trained incrementally, whereas IND can be trained only in batch mode. Classifiers that are batch trained typically have an advantage over those that are trained incrementally: processing the entire training data set when building a classifier may produce a better data model than that produced by incremental training. Therefore, batch training often yields higher accuracy and faster testing.

Table 5 compares MESO results (repeated from Table 4) with those measured for the IND and HDR classifiers. The implementation of IND, written in C, was provided by Buntine, formerly with NASA's Bayesian Learning Group (http://ic.arc.nasa.gov/ic/projects/bayes-group/ind). The HDR implementation, which uses both C and C++, was provided by Weng of the Embodied Intelligence Laboratory at Michigan State University (http://www.cse.msu.edu/ei). MESO is implemented in C++. IND can be used to build a decision tree style classifier using several different algorithms. We tested three different algorithms: CART [44], ID3 [45], and Bayesian [46]. We conducted two sets of experiments with HDR, one using batch training and the other using incremental training.

Let us first compare the MESO results with those of IND. As shown, despite its use of incremental training, MESO accuracy compares favorably with that of all three IND variations, exhibiting higher accuracy in almost all cases. The NC designation indicates that IND could not complete a particular test. Specifically, for the AT&T Faces data set, insufficient memory prevented IND from completing the data set encoding process, which must be done before IND is trained. Somewhat surprisingly, MESO exhibits high accuracy for the Mushroom data set. This data set consists entirely of nominal values, which have no comparative numeric value since they simply indicate characteristics, such as cap shape, by name. IND, like many decision tree algorithms [47], addresses the issue by designating some features as nominal. MESO does not explicitly address nominal values, but still accurately classifies these patterns.

Next, let us consider the training and testing times of MESO relative to those of IND. Although MESO exhibits slower testing times than IND for most data sets, in many cases, MESO spends less time training, which would help to reduce the overhead in acquiring and assimilating new experiences in an online decision maker. Moreover, incremental training as provided by MESO is important to autonomic systems that need to address dynamic environments and changing needs of users.

Finally, let us compare MESO with HDR, which was designed primarily for computer vision tasks. Batch-trained HDR demonstrates slightly higher accuracy than MESO, attributable to HDR's use of discriminant analysis to help select salient features from the training patterns. However, when HDR is trained incrementally, MESO achieves higher accuracy on all eight data sets, including the two image data sets, AT&T Faces and MNIST. Moreover, the training and testing times of MESO are significantly lower than those of HDR in almost all cases. In several cases, the advantage is more than an order of magnitude. Collectively, these results indicate that MESO may be effective in a variety of autonomic applications requiring online decision making.

5 Case Study: Adaptive Error Control

To explore the use of MESO to support learning in adaptive software, we conducted a case study involving adaptive error control. Specifically, we used MESO to implement the decision maker in an audio streaming network application, called XNetApp, that adapts to changes in packet loss rate in a wireless network. XNetApp uses forward error correction (FEC), whereby redundant information is inserted into the data stream, enabling a receiver to correct some losses without contacting the sender for retransmission. In our experimental scenario, depicted in Fig. 10, a stationary workstation transmits an audio data stream to a wireless access point, which forwards the stream to a mobile receiver over the wireless network. As a user roams about the wireless cell and encounters different wireless channel conditions, XNetApp should dynamically adjust the level of FEC in order to maintain a high-quality audio stream. However, XNetApp should also attempt to do so efficiently, that is, it should not consume channel bandwidth unnecessarily.

5.1 Block-Erasure Codes

The FEC method used in this study addresses *erasures* of packets resulting from CRC-based detection of errors at the data link layer. As shown in Fig. 11, an (n, k) *block erasure code* [48] converts k source packets into n encoded packets,

TABLE 5
MESO Accuracy and Training and Test Times When Compared with IND and HDR

Data set	MESO (Incremental)	IND (Batch) CART	IND (Batch) ID3	IND (Batch) Bayesian	HDR (Batch)	HDR (Incremental)
Iris						
Accuracy %	96.1 ±1.4%	92.8 ±0.3%	93.5 ±0.7%	94.2 ±1.1%	96.4 ±1.4%	89.5 ±3.6%
Training (s)	0.0±0.0	0.0±0.6	0.0±0.0	0.01±0.0	0.0±0.0	0.0±0.0
Testing (s)	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
ATT Faces						
Accuracy %	94.0 ±1.4%				94.8 ±1.7%	93.1 ±1.9%
Training (s)	1.9±0.0	NC	NC	NC	9.0±0.2	1.8±0.1
Testing (s)	0.1±0.0				0.3±0.0	0.3±0.0
Mult. Feature						
Accuracy %	94.1 ±0.5%	93.1 ±0.6%	94.2 ±0.2%	94.4 ±1.1%	95.2 ±0.4%	88.8 ±0.5%
Training (s)	1.7±0.0	22.2±0.3	8.6±0.0	19.2±0.2	2.1±0.0	2.5±0.0
Testing (s)	0.07±0.0	0.0±0.0	0.0±0.0	0.0±0.0	2.1±0.0	0.5±0.0
Mushroom						
Accuracy %	100.0 ±0.0%	99.9 ±0.0%	100.0 ±0.0%	100.0 ±0.0%	100.0 ±0.0%	64.6 ±0.6%
Training (s)	0.6±0.0	0.7±0.0	0.0±0.0	0.0±0.0	4.5±0.1	4.2±0.1
Testing (s)	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	1.4±0.0	1.1±0.0
Japanese Vowel						
Accuracy %	91.5 ±0.3%	82.3 ±0.3%	84.2 ±0.3%	84.7 ±0.3%	96.3 ±0.2%	84.9% ±0.8%
Training (s)	0.4±0.0	82.3±0.3	2.6±0.0	7.0±0.6	0.9±0.0	5.0±0.2
Testing (s)	0.1±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.9±0.0	1.3±0.0
Letter						
Accuracy %	90.6 ±0.3%	84.4 ±0.3%	87.9 ±0.1%	88.6 ±0.2%	93.4 ±0.1%	86.2 ±1.0%
Training (s)	1.3±0.0	5.4±0.1	0.9±0.0	3.0±0.0	5.0±0.1	30.8±0.5
Testing (s)	0.2±0.0	0.0±0.0	0.0±0.0	0.1±0.0	0.6±0.0	7.0±0.1
MNIST						
Accuracy %	94.3 ±0.1%	88.3 ±0.1%	88.1 ±0.1%	89.0 ±0.1%	97.4 ±0.0%	91.2 ±0.8%
Training (s)	70.9±1.1	1225.2±48.2	211.9±3.8	565.1±42.4	5887.0±439.0	3997.9±252.4
Testing (s)	6.7±0.2	12.7±1.2	10.9±0.1	10.9±0.1	6007.7±158.2	898.1±55.8
Cover Type						
Accuracy %	96.1 ±0.0%	93.9 ±0.9%	95.2 ±0.2%	94.4 ±0.3%	96.6% †	71.2% †
Training (s)	110.0±0.4	414.4±3.0	51.5±0.2	118.9±5.1	41164.0	52755.3
Testing (s)	18.7±0.1	0.5±0.0	0.6±0.2	1.0±0.3	15148.0	11600.0

All tests began with $\delta = 0.0$ and $c = 0.6$. Executed on a 2GHz Intel Xenon processor with 1.5GB RAM running Linux. All experiments conducted using cross-validation. † The Cover Type data set was not completed for either batch or incremental executions of HDR. Neither was completed due to long execution time requirements.

such that any k of the n encoded packets can be used to reconstruct the k source packets. These codes have gained popularity recently due to an efficient implementation by Rizzo [49]. Each set of n encoded packets is referred to as a *group*. Here, we use only *systematic* (n, k) codes, meaning that the first k packets in a group are identical to the original k data packets. The remaining $n - k$ packets are referred to as *parity* packets.

In earlier studies, our group has investigated several ways that mobile systems can adapt to changing conditions on wireless networks. Examples include adaptable proxies for video streaming [50], adaptive FEC for reliable multicasting [51], several adaptive audio streaming protocols [52], [53], and the design of middleware components whose structure and behavior can be modified at run time in response to dynamic conditions [54]. However, in those approaches, the rules used to govern adaptation were developed in an ad hoc manner as a result of experiments. Here, we investigate whether the system itself can *learn* how to adapt to dynamic conditions.

5.2 Features

In the experiments, 56 environmental features are sensed directly, or calculated from other features, and used as input to the decision-making process. The features are listed in Table 6. The first four features are instantaneous

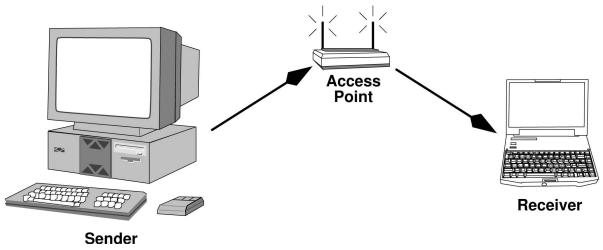


Fig. 10. Physical network configuration used in XNetApp case study.

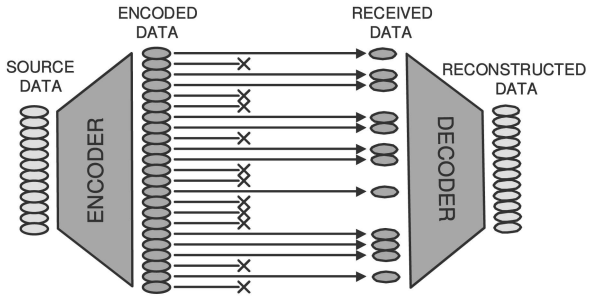


Fig. 11. Operation of FEC based on block erasure codes.

TABLE 6
Features Used for Training and Testing the XNetApp

#	Feature Description
1–4	Instantaneous measurements: bandwidth, perceived packet delay, perceived loss and real loss.
5–32	Time-sampled measurements: median, average, average deviation, standard deviation, skewness, kurtosis and derivative.
33-56	Fourier spectrum of the time-sampled measurements: median, average, average deviation, standard deviation, skewness and kurtosis.

measurements. Perceived features represent the application's viewpoint. That is, perceived packet loss represents the packet loss as observed by the application after error correction, while real packet loss is the number of packets actually dropped by the network prior to error correction. The second group of 28 features is produced by applying seven different metrics (mean, standard deviation, etc.) to each of the four directly measured features as sampled over time. The last group of 24 features is produced by calculating six Fourier spectrums for each of the four directly measured features.

The decision maker's goal is to consider these 56 features and autonomously adapt the system to recover from network packet loss while conserving bandwidth. The adaptation is realized by having the receiving node request the sender to modify the (n, k) settings and change the packet size. The decision maker needs to increase the level of error correction when packet loss rates are high and reduce the level of error correction when packet loss rates are low.

Audio is sampled at 8 KHz using 16-bit samples. Each packet includes a 12-byte application level header containing a sequence number, stream offset, and data length. So, for example, a 32-byte packet contains the header and 10 samples, equivalent to 1.25 milliseconds of audio. We experimented with larger packet sizes and other n,k combinations, but the above values provided sufficient diversity in MESO-based learning and autonomous decision making.

5.3 Imitative Learning

In our experiments, the XNetApp decision maker uses MESO to "remember" user preferences for balancing packet loss with bandwidth consumption. The decision maker gains this knowledge through *imitative learning*. A user shows the XNetApp how to adapt to a rising loss rate by selecting an (n, k) setting with greater redundancy. If the new setting reduces the perceived loss rate to an acceptable level, the user reinforces the new configuration (e.g., by pressing a particular key), and the XNetApp uses MESO to associate the sensed environment and selected (n, k) configuration. Later, when operating autonomously, the decision maker senses current environmental conditions and calculates time-sampled and Fourier features, constructing a pattern. Using this pattern, the XNetApp queries MESO for a system configuration that most likely addresses current conditions. Then, the decision maker emulates the user's actions and adapts the XNetApp, changing the configuration to match that returned from MESO.

5.4 Results

We report results of experiments designed to evaluate the ability of the XNetApp to autonomously balance error control effectiveness and bandwidth consumption. The transmitting station was a 1.5GHz AMD Athlon workstation, and the mobile receiver was a a 500MHz X20 IBM Thinkpad notebook computer. Both systems run the Linux operating system. We report results for two sets of experiments.

The first set of experiments was conducted in a controlled setting, specifically using a wired network and artificially generated packet losses. These experiments were designed to verify that XNetApp could learn to respond accurately to a simple loss model. We trained and tested XNetApp using TCP over a 100Mb wired network, thereby avoiding the effects of spurious errors and overruns of UDP buffers. Packets were dropped at the sender according to a probabilistic loss model, which varied the loss rate from 0.0 to 0.3 in steps of size 0.05, at 15 second intervals. After starting the receiver and sender, the system was trained by having a user select (n, k) values and packet sizes in an attempt to minimize the perceived loss and bandwidth consumption. When a combination satisfying user preferences is found, the XNetApp (receiver) is notified that the current combination is "good" (by pressing the "g" key). Good FEC/packet size combinations and system measurements were then used to train MESO. Training concluded in one hour with MESO storing 34,982 training patterns associated with six FEC code combinations: 32(10,2), 32(8,2), 64(1,1), 64(4,2), 64(6,2), and 64(8,2). In testing, XNetApp collected system measurements and used them to query MESO for the FEC code/packet size combination associated with the most similar set of measurements observed during training.

Figs. 12a and 12b, respectively, show the (artificially generated) network packet loss and the perceived packet loss during the testing phase of the experiment. All changes to error correction are made autonomously by the XNetApp decision maker. Fig. 12c plots the redundancy-ratio defined as $\frac{(n-k)}{n}$, reflecting the changes in FEC (n, k) values corresponding to the loss rates shown in Fig. 12a. For comparison, Fig. 12c also depicts a plot of the optimum redundancy ratio given the FEC codes specified during training. The optimum ratio is computed using the FEC code that provides redundancy greater than or equal to the real loss rate. From these figures, it can be seen that the XNetApp significantly reduces packet loss as perceived by the application by automatically adapting FEC parameters and packet size. Notably, in order to conserve bandwidth, the XNetApp did not simply choose a high (n, k) ratio, but changed parameters to correspond with the changing loss rate.

The second set of experiments were conducted using real packet losses on an 11Mbps 802.11b wireless network. The experimental configuration is shown in Fig. 10. These tests required XNetApp to autonomously balance real packet loss and bandwidth consumption as a user roamed about a

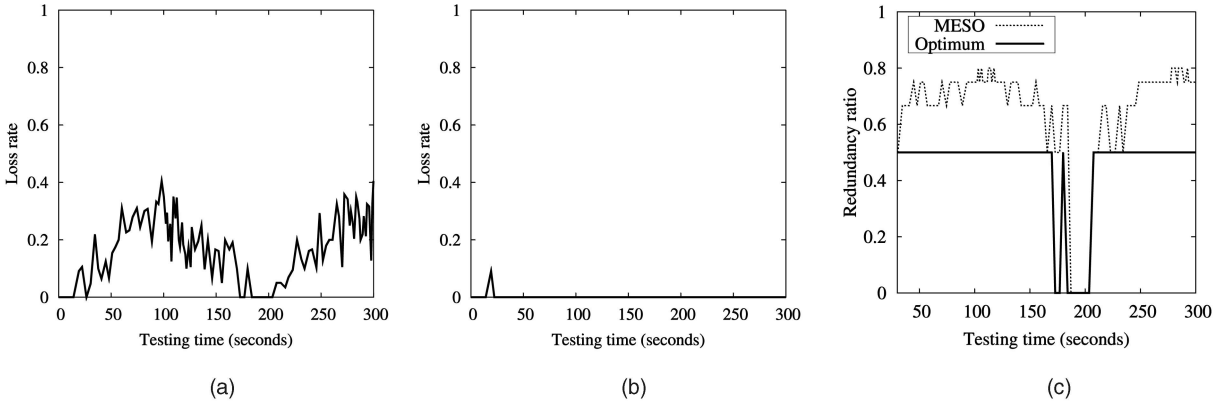


Fig. 12. XNetApp results for artificially generated packet losses. (a) Network packet loss. (b) Perceived packet loss. (c) Redundancy ratio.

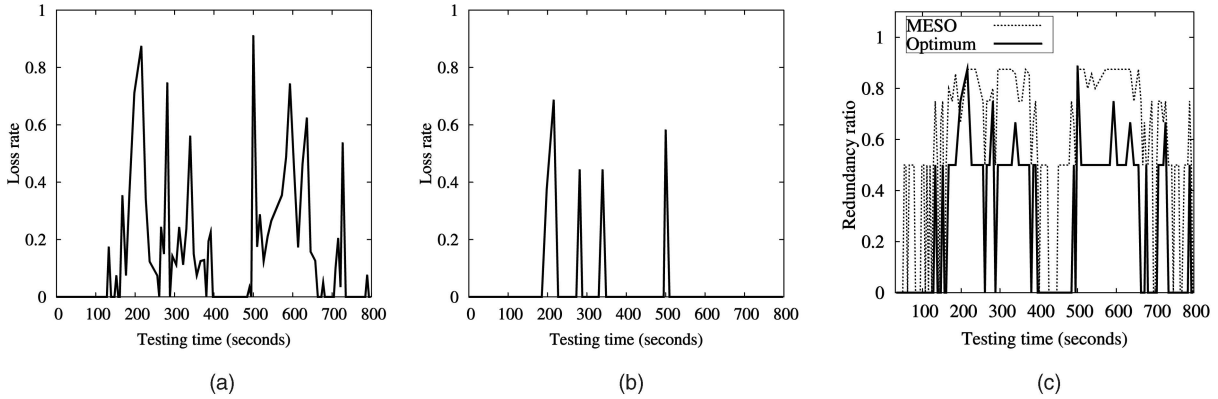


Fig. 13. XNetApp results for real packet losses on a wireless network. (a) Network packet loss. (b) Perceived packet loss. (c) Redundancy ratio.

wireless cell. The XNetApp was trained by a user for one hour using an artificial loss rate that varied from 0.0 to 0.6 in steps of size 0.05 at 15 second intervals. Such a model allowed the XNetApp to be trained for the higher loss rates often found at the periphery of a real wireless cell. Training generated 32,709 training patterns in 10 classes that were used to train MESO for autonomous testing atop a wireless network. Each class "label" is a FEC configuration specifying a (n, k) pair and a packet size. The 10 classes (packet size/FEC code combinations) were:

$$32(10,2)\ 32(12,2)\ 32(14,2)\ 32(16,2)\ 32(18,2)$$
$$32(8,2)\ 64(1,1)\ 64(4,2)\ 64(6,2)\ 64(8,2).$$

In the testing phase, we turned off simulation and enabled the XNetApp to autonomously balance real packet loss and bandwidth consumption. The sender was located on a stationary workstation connected to a wireless access point through a 100Mb hub. A wireless PCMCIA card provided network access to the notebook computer. The UDP/IP multicast protocol was used for transmission of the data stream. Data was collected as a user roamed about a wireless cell carrying a notebook running an XNetApp receiver. Again, all changes to error correction were made autonomously by the XNetApp decision maker. Fig. 13 shows the the results using the same format as in the earlier tests. Under real conditions, XNetApp is able to significantly reduce loss rate as perceived by the application, while conserving bandwidth under good channel conditions.

Table 7 shows results from running cross-validation tests using the data acquired during XNetApp training. This data was produced during training for autonomous XNetApp operation on the real wireless network. This table shows accuracy, with and without compression, helping quantify how well the XNetApp can be expected to imitate a user. The system achieved 94 percent accuracy without compression, and maintained an accuracy level above 87 percent even when data was compressed by more than 90 percent. We regard these results as promising and justifying further study of MESO for online decision making in autonomic systems.

6 Conclusions and Future Directions

We have presented a perceptual memory approach, called MESO, that uses pattern classification and clustering techniques to support online decision making in autonomic systems. We showed that, when used as a pattern classifier, MESO can accurately and quickly classify patterns in several standard data sets, comparing favorably to existing classifiers. We also designed an adaptable framework and implemented an application, XNetApp, that imitatively learns how to make decisions through interaction with a user. XNetApp was successfully trained, using imitative learning, to change the level of error correction while minimizing bandwidth consumption in response to changing network conditions. We postulate that software, such as the XNetApp, that can be trained to make good decisions may simplify the integration of software into new or

TABLE 7
XNetApp Results with and without Compression

Data set	Uncompressed	Means	Spherical	Orthogonal
XNetApp				
Accuracy%	94.1%±0.2%	87.7%±0.2%	92.4%±0.7%	87.3%±0.4%
Compression%	0.0%	91.8%±0.1%	5.8%±0.2%	91.8%±0.14%
Training (s)	33.096±2.123	18.059±0.3	74.705±6.5	19.359±0.582
Testing (s)	1.127±0.016	1.024±0.013	1.119±0.014	1.024±0.010

Data set size is 32,709. Executed on a 2GHz Intel Xenon processor with 1.5GB RAM running Linux. All experiments conducted using cross-validation.

pervasive computing environments. Moreover, a user can teach an application how to meet his or her needs in the face of mobility and novel environments.

In future work, we plan to address the issue of novel experiences with respect to perceptual memory and decision making. Online decision makers may be faced with the uncertainty present in dynamic environments, as new situations are encountered. When a novel pattern of sensed values or a new user action is first encountered, it may initially be considered as an outlier. However, this pattern might also reflect a change in environmental conditions or user preference. We plan to explore the relationship between outliers and novelty in dynamic environments and how novel experience affects the decision making process. One possible approach is to enable MESO to "forget" rarely used patterns or sensitivity spheres, helping both to eliminate the impact of outliers and outdated sensor data on classifier accuracy and to reduce memory and processor consumption during extended online data acquisition. We also intend to explore cases where MESO might overfit the training data, producing a decision boundary that may not generalize well to making decisions in real-world environments. In such situations, if the decision maker can recognize when there is significant uncertainty associated with a "remembered" solution, it may choose to invoke a planning strategy rather than rely on what was remembered.

Further information. A number of related papers and technical reports of the Software Engineering and Network Systems Laboratory can be found at http://www.cse.msu.edu/sens.

ACKNOWLEDGMENTS

The authors would like to thank Juyang Weng, Xiao Huang, and Dave Knoester at Michigan State University for their contributions to this work. This work was supported in part by the US Department of Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by US National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142.

REFERENCES

[1] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H. Cheng, "Composing Adaptive Software," *Computer,* vol. 37, pp. 56-64, July 2004.
[2] *Proc. Second Int'l Conf. Autonomic Computing (ICAC),* June 2005.
[3] *Proc. Distributed Auto-Adaptive and Reconfigurable Systems Workshop (DARES), held in conjunction with the 24th Int'l Conf. Distributed Computing Systems (ICDCS),* Mar. 2004.
[4] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* pp. 41-50, Jan. 2003.
[5] J.M. Fuster, *Memory in the Cerebral Cortex: An Empirical Approach to Neural Networks in the Human and Nonhuman Primate.* The MIT Press, 1995.
[6] S. Franklin, "Perceptual Memory and Learning: Recognizing, Categorizing and Relating," *Proc. Developmental Robotics AAAI Spring Symp.,* Mar. 2005.
[7] T. Jebara and A. Pentland, "Statistical Imitative Learning from Perceptual Data," *Proc. Second Int'l Conf. Development and Learning,* pp. 191-196, June 2002.
[8] R. Amit and M. Matarić, "Learning Movement Sequences from Demonstration," *Proc. Second Int'l Conf. Development and Learning,* pp. 165-171, June 2002.
[9] E.P. Kasten and P.K. McKinley, "MESO: Perceptual Memory to Support Online Learning in Adaptive Software," *Proc. Third Int'l Conf. Development and Learning (ICDL '04),* Oct. 2004.
[10] W. Buntine, "Tree Classification Software," *Proc. Third Nat'l Technology Transfer Conf. and Exposition,* Dec. 1992.
[11] W.-S. Hwang and J. Weng, "Hierarchical Discriminant Regression," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 11, Nov. 2000.
[12] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification,* second ed. John Wiley and Sons, 2001.
[13] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Pearson Education, Inc., 2006.
[14] S. Murthy, S. Kasif, and S. Salzberg, "A System for Induction of Oblique Decision Trees," *J. Artificial Intelligence Research (JAIR),* vol. 2, pp. 1-32, 1994.
[15] C.C. Aggarwal, C. Procopiuc, J.L. Wolf, P.S. Yu, and J.S. Park, "Fast Algorithms for Projected Clustering," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 61-72, June 1999.
[16] S. Kumar, J. Ghosh, and M.M. Crawford, "Hierarchical Fusion of Multiple Classifiers for Hyperspectral Data Analysis," *Pattern Analysis and Applications,* vol. 5, pp. 210-220, 2002.
[17] J. Tantrum, A. Murua, and W. Stuetzle, "Assessment and Pruning of Hierarchical Model Based Clustering," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 2003.
[18] J. Tantrum, A. Murua, and W. Stuetzle, "Hierarchical Model-Based Clustering of Large Datasets through Fractionation and Refractionation," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 183-190, July 2002.
[19] H. Yu, J. Yang, and J. Han, "Classifying Large Data Sets Using SVMs with Hierarchical Clusters," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 306-315, Aug. 2003.
[20] A. Kalton, P. Langley, K. Wagstaff, and J. Yoo, "Generalized Clustering, Supervised Learning, and Data Assignment," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 299-304, Aug. 2001.
[21] J. Kivinen, A.J. Smola, and R.C. Williamson, "Online Learning with Kernels," *Proc. Advances in Neural Information Processing Systems (NIPS),* 2002.
[22] K. Crammer, J. Kandola, and Y. Singer, "Online Classification on a Budget," *Proc. Advances in Neural Information Processing Systems (NIPS),* 2003.
[23] C. Gupta and R. Grossman, "GenIc: A Single Pass Generalized Incremental Algorithm for Clustering," *Proc. SIAM Int'l Conf. Data Mining,* Apr. 2004.
[24] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB '97),* pp. 426-435, Aug. 1997.

[25] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. 1996 ACM SIGMOD Int'l Conf. Management of Data,* pp. 103-104, June 1996.

[26] M.M. Breunig, H.-P. Kriegal, P. Kröger, and J. Sander, "Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data,* May 2001.

[27] Y.A. Ivanov and B.M. Blumberg, "Developmental Learning of Memory-Based Perceptual Models," *Proc. Second Int'l Conf. Development and Learning,* pp. 165-171, June 2002.

[28] M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic, Internet Services," *Proc. Int'l Conf. Dependable Systems and Networks (IPDS Track),* 2002.

[29] P. Geurts, I.E. Khayat, and G. Leduc, "A Machine Learning Approach to Improve Congestion Control over Wireless Computer Networks," *Proc. Fourth IEEE Conf. Data Mining (ICDM '04),* pp. 383-386, Nov. 2004.

[30] J.A. Hartigan, *Clustering Algorithms.* John Wiley and Sons, 1975.

[31] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computer Surveys,* vol. 31, pp. 264-323, Sept. 1999.

[32] C.L. Blake and C.J. Merz, "UCI Repository of Machine Learning Databases," http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[33] S. Hettich and S.D. Bay, "UCI KDD Archive," http://kdd.ics.uci.edu, 1999.

[34] F. Samaria and A. Harter, "Parameterisation of a Stochastic Model for Human Face Identification," *Proc. Second IEEE Workshop Applications of Computer Vision,* Dec. 1994.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE,* vol. 86, pp. 2278-2324, Nov. 1998.

[36] R.A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics,* vol. 7, pp. 179-188, 1936.

[37] M. van Breukelen, R.P.W. Duin, D.M.J. Tax, and J.E. den Hartog, "Handwritten Digit Recognition by Combined Classifiers," *Kybernetika,* vol. 34, no. 4, pp. 381-386, 1998.

[38] A.K. Jain, R.P.W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, pp. 4-37, Jan. 2000.

[39] P.W. Frey and D.J. Slate, "Letter Recognition Using Holland-Style Adaptive Classifiers," *Machine Learning,* vol. 6, Mar. 1991.

[40] J.S. Schlimmer, "Concept Acquisition through Representational Adjustment," PhD thesis, Dept. of Information and Computer Science, Univ. of California, Irvine, 1987.

[41] M. Kudo, J. Toyama, and M. Shimbo, "Multidimensional Curve Classification Using Passing-Through Regions," *Pattern Recognition Letters,* vol. 20, pp. 1103-1111, 1999.

[42] J.A. Blackard and D.J. Dean, "Comparative Accuracies of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables," *Proc. Second Southern Forestry GIS Conf.,* pp. 189-199, 1998.

[43] J. Weng and W.-S. Hwang, "An Incremental Learning Algorithm with Automatically Derived Discriminating Features," *Proc. Asian Conf. Computer Vision,* pp. 426-431, Jan. 2000.

[44] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees.* Chapman and Hall, 1984.

[45] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, pp. 81-106, 1986.

[46] W.L. Buntine, "Decision Tree Induction Systems: A Bayesian Analysis," *Proc. Third Conf. Uncertainty in Artificial Intelligence,* pp. 109-128, July 1987.

[47] S.K. Murthy, "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey," *Data Mining and Knowledge Discovery,* vol. 2, no. 4, pp. 345-389, 1998.

[48] A.J. McAuley, "Reliable Broadband Communications Using Burst Erasure Correcting Code," *Proc. ACM SIGCOMM,* pp. 287-306, Sept. 1990.

[49] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *ACM Computer Comm. Rev.,* vol. 27, pp. 24-36, Apr. 1997.

[50] P. Ge and P.K. McKinley, "Leader-Driven Multicast for Video Streaming on Wireless LANs," *Proc. IEEE Int'l Conf. Networking,* Aug. 2002.

[51] P.K. McKinley, C. Tang, and A.P. Mani, "A Study of Adaptive Forward Error Correction for Wireless Collaborative Computing," *IEEE Trans. Parallel and Distributed Systems,* Sept. 2002.

[52] P.K. McKinley, U.I. Padmanabhan, N. Ancha, and S.M. Sadjadi, "Composable Proxy Services to Support Collaboration on the Mobile Internet," *IEEE Trans. Computers,* special issue on wireless Internet, pp. 713-726, June 2003.

[53] Z. Zhou, P.K. McKinley, and S.M. Sadjadi, "On Quality-of-Service and Energy Consumption Tradeoffs in fec-Enabled Audio Streaming," *Proc. 12th IEEE Int'l Workshop Quality of Service (IWQoS '04),* June 2004.

[54] S.M. Sadjadi, P.K. McKinely, and E.P. Kasten, "Architecture and Operation of an Adaptable Communication Substrate," *Proc. Ninth Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '03),* May 2003.

Eric P. Kasten received the BS degree in mathematics and computer science from Central Michigan University in 1989 and the MS degree in computer science from Michigan State University in 1997. He is currently a PhD candidate in the Department of Computer Science and Engineering, and a software developer in the National Superconducting Cyclotron Laboratory, both at Michigan State University. His current research interests include autonomic computing, dynamic system adaptation, and data stream processing and mining in support of ecosensing and adaptive mobile computing. He is a member of the IEEE and the IEEE Computer Society.

Philip K. McKinley received the BS degree in mathematics and computer science from Iowa State University in 1982, the MS degree in computer science from Purdue University in 1983, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1989. Dr. McKinley is currently a professor of computer science and engineering at Michigan State University. He was previously a member of technical staff at Bell Laboratories. He has served as an associate editor for the *IEEE Transactions on Parallel and Distributed Systems* and was cochair of the program committee for the 2003 IEEE International Conference on Distributed Computing Systems. His current research interests include self-adaptive software, digital evolution, mobile computing, and group communication protocols. He is a member of the IEEE and the IEEE Computer Society.