Progressive Partial Memory Learning

A Dissertation Submitted to the Graduate Faculty
of George Mason University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy,
Information Technology

By

Marcus A. Maloof

Bachelor of Science
University of Georgia, 1989

Master of Science
University of Georgia, 1992

Director: Ryszard S. Michalski, PRC Chaired Professor
Departments of Computer Science and Systems Engineering

Fall 1996
George Mason University
Fairfax, Virginia

DEDICATION

To Giddie: Nassir “Louis” Ackle Maloof

1894–1994

ACKNOWLEDGMENTS

Many friends and colleagues have contributed significantly to this work. I thank my Dissertation Director, Ryszard Michalski, for overseeing this research and contributing ideas, while providing the environment and freedom to develop and follow my own. I appreciate the support, ideas, and comments of my committee members: Ophir Frieder, Gheorghe Tecuci, and David Schum. Thanks to Bob Holt and Wayne Gray of the Department of Psychology for discussions about experimental design and analysis. Thanks to Gerhard Widmer of the Austrian Research Institute for Artificial Intelligence for running the experiments and providing the data points necessary to produce Figure 25. Thanks to Jayshree Sarma for pointers to papers on concept change in the genetic algorithm literature. Thanks to Zoran Duric for helping with the mathematical derivation in Section 4.4 and for many educational discussions about computer vision, especially as it relates to machine learning. Thanks to Azriel Rosenfeld of the Center for Automation Research, University of Maryland, College Park, for discussions about modern day computer vision systems, especially as they relate to the blasting cap detection problem. Thanks to Gary Jackson of Psychological Assessment Resources, Inc. for many fruitful discussions about machine learning, statistical validation and for supporting me in all aspects of this pursuit. Eric Bloedorn provided the email learning agent data set and has been a friend and close collaborator throughout this process. His discussions helped shape many of the ideas presented here. Thank you. Thanks to Ken Kaufman for invaluable insights into AQ internals. Thanks to other members of the Machine Learning and Inference Laboratory, past and present, for help along the way: Jerzy Bala, Janusz Wnek, Qi Zhang, and Ibrahim

Imam. Thanks to the School of Information Technology and Engineering for the support of three doctoral fellowship awards. Finally, I want to thank my family and friends for their support, understanding, and counsel throughout this process.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research is supported in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research, and Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, in part by the Office of Naval Research under Grant No. N00014-91-J-1351, and in part by the National Science Foundation under Grants No. IRI-9020266 and DMI-9496192.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

PROGRESSIVE PARTIAL MEMORY LEARNING
Marcus A. Maloof
George Mason University, 1996
Dissertation Director: Ryszard S. Michalski

A learning methodology called Progressive Partial Memory Learning (PPML) is presented. PPML takes a partial memory approach to progressive inductive learning problems in which training examples are distributed over time. The partial memory approach retains and uses representative examples and induced concept descriptions for future learning. Representative examples are those training examples that maximally expand and constrain concept descriptions in the representation space. Mechanisms such as the selection of representative examples, forgetting, and aging allow the system to efficiently learn concepts over time and to track changing concepts through a representation space. Key components of the methodology are implemented in an experimental system called AQ-Partial Memory (AQ-PM), which is based on the AQ15c inductive learning system. AQ-PM is experimentally validated against a baseline AQ15c learning algorithm using both synthetic and real-world data sets. Synthetic data sets include problems in which concepts change or drift in the representation space. Real-world problems include applications to dynamic knowledge bases (i.e., computer intrusion detection), intelligent agents (i.e., email sorting), and computer vision (i.e., blasting cap detection in x-ray images). Results demonstrate that the methodology is able to perform well on a variety of problems. Specifically, the method considerably improves memory requirements and learning time with slight decreases in predictive accuracy when compared to the baseline learner.

Furthermore, the method is also able to track concept drift. Results are presented for the STAGGER concepts in which AQ-PM achieves predictive accuracies comparable to the FLORA systems, but requires much less memory. Comparisons are made to the AQ11 and GEM incremental learning systems using the blasting cap detection and computer intrusion detection problems in which they learned more predictive, but more complex concept descriptions than AQ-PM. Partial memory learning is also conducted using the incremental learning algorithms of AQ11 and GEM for the blasting cap detection and computer intrusion detection problems. Results for the partial memory versions of these learners, when compared to the unmodified versions, show slight decreases in predictive accuracy and concept complexity with considerable decreases in learning time.

Chapter 1: Introduction

Two years ago during a research meeting, we were discussing an incident in which a hacker broke into a university administrator's account. The hacker's activity was characterized by late night logins and intense usage of computing resources. In short, the hacker was doing things the true owner of the account would have never done. As a student in machine learning, it seemed natural to use machine learning to develop behavioral profiles of the various users of a computing system, and if a given user did not match a historical profile, then either the actual user was not the true user, or the system needed more training.

After gathering three weeks of audit records, the odyssey of finding an appropriate representation space for learning began. It took a while — about a year of repeatedly leaving the problem and coming back to it — but once it was found, the learning results were quite good. Naturally, the next step was to make several comparisons among various learning methods and report how well each did on this problem.

Yet, the engineering question is, How would one build a system to do intrusion detection? The system must constantly learn and classify. It must be able to incorporate feedback from the system administrator in the event of an erroneous classification. It should use little computing and storage resources, meaning fast learning and recognition and low memory requirements, since it would be nice not to have to buy a computing system to protect a computing system.

Learning over time means progressive learning (Michalski 1996). By progressive learning, we mean a learning process of building hypotheses from consecutively obtained

sets of training data. Learning over time is an important special case of progressive learning. After looking at the current literature on intrusion detection and on systems that learn over time, under the influence of current course work, readings, and lengthy chats with colleagues, a design for a progressive learning system began to materialize. Initially, experimentation with the progressive partial memory learning system was not fully automated, which provided the opportunity to truly look at what was happening with the problem. As time progressed and the system received more and more training examples, the examples retained in earlier steps of learning were either in conflict with examples retained in later learning steps or produced rules covering few examples. It was apparent that what was expressed by the earlier training examples was not present in the later examples. In other words, the concepts were changing.

Other applications, not just the computer intrusion detection problem, have similar properties. These applications include intelligent agents (Maes 1994) and a variety of vision problems, such as active vision (Aloimonos et al. 1988) and learning to track objects in video sequences (Blake et al. 1995). Thus the research question is, What are the properties of a learning methodology that would allow us to build learning systems for this class of problems?

We present a learning methodology called Progressive Partial Memory Learning, or PPML (Maloof and Michalski 1995c). PPML takes a partial memory approach to progressive learning in which training examples are distributed over time. The partial memory approach retains and uses representative examples and induced concept descriptions for future learning. Representative examples are those training examples that maximally expand and constrain concept descriptions in the representation space. Mechanisms such as the selection of representative examples, forgetting, and aging allow the system to efficiently learn concepts over time and to track changing concepts through a representation space.

Key components of the methodology are implemented in an experimental system called AQ-Partial Memory (AQ-PM), which is based on the AQ15c inductive learning system (Wnek et al. 1995). AQ-PM is experimentally validated against a baseline AQ15c learning algorithm using both synthetic and real-world data sets. Real-world problems include applications to computer intrusion detection (Maloof and Michalski 1995a), email sorting (Bloedorn and Michalski 1996), and detecting blasting caps in x-ray images (Maloof and Michalski 1995d, 1996; Maloof et al. 1996). Experimental results demonstrate that the methodology is able to perform well on a variety of problems. Specifically, the method considerably improves memory requirements and learning time with slight decreases in predictive accuracy when compared to the AQ15c baseline learner. The baseline system learned as much as 113% slower, maintained as much as 256% more data, and performed as little as 4% better on testing data than the AQ partial memory learner.

A synthetic problem is considered in which concepts change or drift in the representation space. Experimental results are presented for the STAGGER concepts (Schlimmer and Granger 1986) in which AQ-PM achieves predictive accuracies comparable to the FLORA systems (Widmer and Kubat 1996), with the FLORA systems requiring, on average, 116% more memory.

Comparisons are also made to two incremental learning versions of the AQ algorithm (Michalski 1969): AQ11 (Michalski and Larson 1983), for no memory incremental learning, and GEM (Reinke and Michalski 1988), for full memory learning. For the blasting cap detection problem and the computer intrusion detection problem, experimental results show that AQ11 and GEM learned more predictive (3–7% more predictive), but more complex (as much as 187% more selectors) concept descriptions than AQ-PM.

Finally, progressive partial memory learning is also conducted using the incremental learning algorithms of AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988) for the blasting caps detection problem and the computer intrusion detection problem. Results for the unmodified versions of these learners, when compared to the partial memory versions, show slight increases in predictive accuracy (1–4%), considerable increases in learning time (as much as 485%), and slight increases in concept complexity (as much as 50% more complex rules). The partial memory approach also reduced or eliminated fluctuations in concept complexity of the AQ11 incremental learner, a problem recognized by other researchers (O'Rorke 1982; Becker 1985).

Chapter 2: Problem Statement

The research problem addressed by this dissertation is the inductive learning of concepts from examples distributed over time. A related problem is that of inductively learning a symbolic concept description of a changing concept. It is assumed that change occurs over time, so training examples are distributed over time. In general, a learner cannot empirically differentiate between these two problems because of the uncertainty of inductive inference. That is, because inductive inference is uncertain, without assumptions, a learner cannot determine if it is receiving more information about a static concept or about a changing concept.

The study of these problems is important and relevant for several reasons. From an applied perspective, several applications, such as dynamic knowledge-bases and intelligent agents, operate in a dynamic and changing environment. Learning systems for these applications must be able to learn quickly with low memory requirements. Furthermore, they must learn changing concepts. Assume, for example, we have an email sorting agent that "watches over the shoulder" of a user associating attributes of various emails (e.g., sender and subject keywords) with user actions (e.g., such as reading and deleting). After a period of training and when the agent can reliably predict its user's actions, the agent may request to automate various email sorting tasks for the user. If the user accepts the agent's request, then the agent may automatically prioritize the user's mail queue, delete unwanted junk emails, and the like.

If the user is a university professor, then the concept of which emails are important is likely to change from semester to semester. That is, the emails from students currently

enrolled in the professor's class are likely to be more important than those emails from students who are not currently enrolled. Yet, when the semester changes, although we still have the concept of important emails, the concept has changed because the students in the professor's class are no longer the same. A learning system for this application must learn over time, since emails are continually arriving, and learn a changing concept, since the concept itself changes.

From a theoretical perspective, partial memory models are interesting because we can investigate the rates of growth for the number of training examples maintained during learning. Other issues relate to the compromises and gains in predictive accuracy, learning times, and concept complexity when taking a partial memory approach. One can also study the change in these measures as different schemes of computing partial memory are employed in which differing portions of the original training set are retained for learning.

In the AQ legacy of learning systems, O'Rorke (1982) noted that AQ11 (Michalski and Larson 1983), a no memory incremental learning system, exhibited cyclic instability in which there would be wide and unwarranted variations in concept complexity from one learning step to the next. One empirical observation made in this dissertation is that maintaining a set of representative examples eliminates this instability of concept complexity.

2.1 Assumptions

The methodology of progressive partial memory learning makes few assumptions about the type of inductive learner involved. It is necessary, however, for the learner to form concept descriptions so that representative examples can be found. Naturally, the specific way in which representative examples are found will vary among learning systems, depending on their concept representation, but the idea of representative examples is not

specific to any one learning system. Furthermore, mechanisms of aging, forgetting, and inductive support are also non-specific.

The methodology assumes a static representation space; that is, constructive induction (Michalski 1980) is not considered. The methodology also assumes a stationary context (cf. Widmer 1996a). Note however that both constructive induction and non-stationary contexts would be fruitful areas for further research.

Assumptions are also made with respect to the implementation of the experimental system, AQ-PM. By using AQ15c (Wnek et al. 1995) as the underlying learning system, we assume that training examples can be represented in a discrete attributional representation space and that concepts can be represented by one or more axis-parallel hyper-rectangles. That is, it is assumed that domain concepts can be represented by a disjunction of conditions involving attributes but not relations. This is our representational bias. Consequently, the system cannot learn structural or recursive concepts. Noise is not considered since other studies have dealt with this problem (Aha et al. 1991; Widmer and Kubat 1996).

With regard to experimental design, the assumption is made for the problems involving real-world data sets that training examples, while distributed over time, are not dependent with respect to time. This assumption is made for practical reasons. By making this assumption, the data sets can be partitioned randomly over time to simulate a time-distributed problem. For each of these partitions, we can then use a validation methodology suggested by Weiss and Kulikowski (1992). Ultimately, this assumption reduces our data requirements. If we were not able to make this assumption, then gathering experimental data would be impractical, if not impossible. For example, consider the computer intrusion detection domain as a learning problem (Maloof and Michalski 1995a). Given the data set used in the study and based on the minimum number of examples per class, we should use a validation methodology of 100 iterations of 2-fold

cross validation. If we cannot make the time independence assumption, we would have to collect data from 100 independent periods of time. The original data set for this problem was collected over a three week period. Thus, we would have to collect data for 300 independent weeks of activity. Similar collection periods would be required for the intelligent agent and computer vision domains. Clearly, this is an unrealistic data collection requirement, hence the assumption.

This assumption, however, is not made for the synthetic data set involving changing concepts. The STAGGER concepts (Schlimmer and Granger 1986) were generated by an algorithm, so data collection is not a problem. Furthermore, the concepts for the synthetic data experiment were constructed to be time dependent.

2.2 Representation Language

A representation language is needed to express algorithms and concept descriptions. For this purpose we will use a simplified version of the Variable-Valued Logic System 1 (VL_1), as described by Michalski (1973), to represent training examples, representative examples, domain knowledge, and concept descriptions. VL_1 decision rules are of the form:

D_i <:: C_j for $i = 1..n$, $j = 1..m$

where

D_i is the *decision* part of the rule and assigns a class label to a decision variable,

C_j is the *condition* part of the rule (i.e., a *complex*), and consists of a conjunction of elementary conditions, and

<:: is the *decision assignment operator* and is logically equivalent to implication.

An elementary condition, or a *selector*, is of the form:

'[' <*referee*> <*relation*> <*referent*> ']'

where

<*referee*> is a member of the finite set of attributes,

<*relation*> is the relational operator =, and

<*referent*> is a subset of the domain of <*referee*>.

A referent is of the form:

<*referent*> ::= <*value*> [<*list*> | <*range*>]

<*list*> ::= '∨' <*value*> [<*list*> | <*range*>]

<*range*> ::= '..' <*value*> [<*list*>]

where <*value*> is from the domain of <*referee*>. For example, [length = 2mm], [color = red ∨ blue], and [area = 1..5 ∨ 7 pixels] are elementary conditions.

In a strict matching mode, an elementary condition is satisfied by an object, if the value of the attribute stated in the condition for this object satisfies the <*relation*> between the <*referee*> and the <*referent*>. A complex is true if all of its selectors are true. A decision variable is assigned its associated class label if its complex is true.

Training examples and representative examples are represented using decision rules under the restrictions that a selector must be present for each attribute in the finite set of attributes and that the cardinality of the referent is singular, i.e., consists of one value from the domain of the attribute. See (Michalski 1973) for a complete description of VL_1.

Chapter 3: Related Research

3.1 Progressive Learning

Progressive learning is a process in which hypotheses are formed from consecutive sets of training data (Michalski 1996). Incremental learning, or on-line learning (Littlestone and Warmuth 1994), and temporal batch learning are types of progressive learning. Incremental learning is a process of concept description modification using newly obtained training examples. Temporal batch learning is a process of re-learning from some or all of the past training examples with any newly obtained training examples.

Progressive learning differs from atemporal batch learning in that it is assumed that training examples are distributed over time. There are many motivations for making this assumption. The obvious one is that the particular problem to which inductive learning is applied necessitates learning over time. Such problems include computer intrusion detection (Denning 1987), intelligent agents (Maes 1994), and robot learning (Salganicoff et al. 1996).

A second motivation is that the computational demands of the domain exceed current computational capabilities. Large training sets can be divided and processed incrementally providing gains in processing speed, since we need not process training examples already covered by current hypotheses, and in memory requirements, since we need not load the entire data set into memory. Early work in incremental learning stemmed from this motivation (Michalski and Larson, 1978, 1983; Reinke and Michalski 1988). Others have observed that incremental learning can lead to improved predictive accuracy and smaller concepts versus batch learning (Utgoff 1989). Finally, some have been

motivated by more cognitive ambitions, since humans and other biological organisms are inherently incremental learners (Feigenbaum 1963).

Michalski (1985) observed that learning over time can be carried out using two mechanisms: revolutionary or evolutionary. With the former case, old concepts are discarded and new concepts are induced from a set of training examples. AQ-PM (Maloof and Michalski 1995c) is an example of a system that operates in a revolutionary, or temporal batch mode. With the latter case, existing concepts are modified in light of new training examples. AQ11 (Michalski and Larson 1983), STAGGER (Schlimmer 1987b), and Winnow (Littlestone 1991) are examples of systems that take an evolutionary approach. GEM (Reinke and Michalski 1988) and CAP (Mitchell et al. 1994) are examples of hybrid incremental learning systems. GEM uses a set of training examples to form an initial set of concepts, and then specializes or generalizes those concepts using new training examples. CAP, on the other hand, learns new rules from new training examples, and integrates these new rules into the existing set of rules, deleting obsolete rules, if necessary.

Progressive learning systems must have a memory model, which determines how the system treats past training examples (Reinke and Michalski 1988). With a no memory model, all training examples are forgotten after they are used to update the system's concepts. AQ11 (Michalski and Larson 1983), STAGGER (Schlimmer 1987b), DISCIPLE (Tecuci and Kodratoff 1990), and Winnow (Littlestone 1991) are systems that use a no memory model. With a partial memory model, some of the past training examples are retained. IB2 (Aha et al. 1991) is one such system that use a partial memory model. With a full memory model, all past training examples are kept. HILLARY (Iba et al. 1988), GEM (Reinke and Michalski 1988), and IB1 (Aha et al., 1991) are examples of systems that use a full memory model. Table 1 lists many of the progressive learning systems developed over the years classified by representation scheme and memory model.

Table 1: Selection of past progressive learning research.

System/Researcher(s)	Representation	Memory Model
EPAM (Feigenbaum 1963)	Decision Trees	No
Arch (Winston 1975)	Semantic Nets	No
AQ11 (Michalski and Larson 1978, 1983)	Decision Rules	No
GEM (Reinke 1984)	Decision Rules	Full
Split (Becker 1985)	Decision Rules	Full
AQ15 (Hong et al. 1986)	Decision Rules	Full
ID4 (Schlimmer and Fisher 1986)	Decision Trees	No
INDUCE 4 (Mehler et al. 1986)	Decision Rules	Full
PRG (Lee and Ray 1986)	Decision Rules	No
UNIMEM (Lebowitz 1986)	Decision Trees	No
COBWEB (Fisher 1987)	Hierarchy	No
LAIR (Wantanabe and Elio 1987)	Decision Rules	Partial
STAGGER (Schlimmer 1987a)	Multistrategy	No
Gross (1988)	Decision Rules	No
HILLARY (Iba et al. 1988)	Decision Rules	Full
ID5 (Utgoff 1988)	Decision Trees	No
CLASSIT (Gennari et al. 1989)	Hierarchy	No
Incremental ILS (Widmer 1989)	Version Spaces	No
Weighted Majority (Littlestone 1989)	Perceptron	No
Winnow (Littlestone 1989)	Perceptron	No
DISCIPLE (Tecuci & Kodratoff 1990)	Version Spaces	No
IB1 (Aha et al. 1991)	Instance-based	Full
IB2 (Aha et al. 1991)	Instance-based	Partial
IB3 (Aha et al. 1991)	Instance-based	Partial
Bhandaru and Murty (1991)	Decision Trees	Full
Dasgupta and McGregor (1992)	Genetic Algorithm	No
FAVORIT (Krizakova and Kubat 1992)	Hierarchy	No
Grefenstette (1992)	Genetic Algorithm	No
Lim et al. (1992)	Connectionist	Partial
PRAX (Bala 1992)	Decision Rules	No
PRORULES (Pitas et al. 1992)	Decision Rules	No
Weiss and Kulikowski (1992)	Connectionist	Full
Cobb and Grefenstette (1993)	Genetic Algorithm	No
DARLING (Salganicoff 1993)	Decision Trees	No
YAILS (Torgo 1993)	Decision Rules	No
CAP (Mitchell et al. 1994)	Decision Trees	No
IVSM (Hirsh 1994)	Version Spaces	Partial
IA-EBL (Cohen 1994)	Decision Rules	No
IVS (Sablon et al. 1994)	Version Spaces	No
EBNN (Thrun and Mitchell 1995)	Multistrategy	No
HIERARCH (Nevins 1995)	Hierarchy	No
VOTE (Clyde et al. 1995)	Weighted IBL	Full
FLORA (Widmer and Kubat 1996)	Decision Rules	Partial
MetaL(B) (Widmer 1996a)	Naive Bayes	Partial
MetaL(IB) (Widmer 1996a)	Instance-based	Partial

3.2 Partial Memory Models

Little research has been conducted in an effort to better understand the performance gains and compromises by progressively learning using different memory models. The majority of progressive learning systems are incremental learning systems that use a no memory model, while most others use a full memory model. IB2 and IB3 (Aha et al. 1992) appear to be two of the first systems to use a partial memory model, but these learning systems do not form generalized concept descriptions. New training examples are kept only if they are misclassified by the system. IB3 uses a similar scheme, but uses additional processes to filter noisy examples.

LAIR (Wantanabe and Elio 1987) uses a minimal partial memory model in which only the last positive example is kept. Incremental learning uses the last positive training example and the new training example when updating the current set of concepts by applying an inductive generalization rule (i.e., climbing a generalization hierarchy).

The FLORA learning systems (Widmer and Kubat 1996), MetaL(B), and MetaL(IB) (Widmer 1996a) use a partial memory model in which the most recently seen training examples are kept. The quantity of kept examples varies based on the size of an adaptive forgetting window. Although these examples are kept, once the examples are used for learning when they first arrive to the system, they are not used in subsequent inductive learning processes. Rather, retained positive and negative examples are used to forget and manage the use of learned concept descriptions.

3.3 Representative Examples

Two studies, in particular, discuss representative examples. The first, by Michalski and Larson (1988), uses incremental learning as a method for processing large volumes of data. Representative examples are selected and used for learning to reduce processing requirements. In this approach, training examples are ranked as a function of their

syntactic distance from a standard reference point in *n*-dimensional space. The range of distances is uniformly scaled into a set of *r* bins, which will likely contain different numbers of examples. The set of representative examples is computed by taking the training examples in each of the *r* bins and finding those examples that are maximally distant from each other. Incremental learning is performed on each reduced bin. The important aspects of this approach is that it is a pre-learning process and representative examples are selected by a syntactic distance measure.

The second study, by Kibler and Aha (1987), discusses representative examples in the context of memory-based or exemplar-based learning. Two algorithms are discussed for selecting a representative set of examples: the growth algorithm and the shrink algorithm. With former algorithm, during learning (i.e., example memorization), only those misclassified examples are kept. This process results in ordering effects whereby the order in which examples arrive affects classification performance. Note that the growth algorithm is used in IB2 and IB3 (Aha et al. 1991).

Regarding the latter algorithm, the shrink algorithm, all training examples are memorized. Then, each training example is tested to see if it would be classified correctly by the remaining examples. If it is classified correctly, it is removed; otherwise, it is retained. As with the growth algorithm, the shrink algorithm is susceptible to ordering effects.

Experimental results are presented using the thyroid data set with a standard memory-based learning algorithm (i.e., the proximity algorithm) as a control and comparing using predictive accuracy, memory requirements, and learning time. Although the growth algorithm stored slightly more training examples than the shrink algorithm, the growth algorithm had higher predictive accuracies for this data set. Furthermore, both algorithms achieved predictive accuracies less than the control algorithm. Computational cost was evaluated formally. If *n* is the total number of training examples, and *m* is the

number of retained training examples, then the control algorithm has a constant update cost of $O(1)$, the growth algorithm requires $O(mn)$, and the shrink algorithm requires $O(n^2)$.

For Kibler and Aha's work, the important point is that representative example selection is a post-learning process. Also, memory-based learning techniques do not form generalized concept descriptions, and selection of representative examples produces ordering effects.

3.4 Concept Change

If a concept changes, then a natural assumption is that this change, whether gradual or abrupt, occurs over time. Evidence of the change in a concept will be communicated by training examples of that concept, which are distributed over time. So not only is a progressive learning system required, but one is required that is capable of learning a concept or function that is changing, evolving (Maloof and Michalski 1995c), shifting (Littlestone and Warmuth 1994), drifting (Schlimmer and Granger 1986), or is non-stationary (Grefenstette 1992; Salganicoff 1993).

STAGGER (Schlimmer 1987a, b) is a multistrategy incremental learning system for coping with changing concepts. STAGGER uses a connectionist representation scheme using nodes to represent Boolean attributes (e.g., color = green) and Bayesian-weighted connections to associate attribute nodes to a concept node. STAGGER learns and tracks changing concepts by adding new attribute nodes or adjusting the connection weights for the concept's connections.

The Weighted-Majority and Winnow algorithms (Littlestone 1989), although studied formally for mistake bounds (i.e., the number of mistakes made during training), were proposed as learning algorithms for tracking concept change (Littlestone and Warmuth 1994). The basic idea behind these algorithms is that a classification problem can be cast into a smaller subset of a representation space that is linearly separable. The

algorithms thus generate all possible couplings, for example, of the attributes, producing some number of linear classifiers, which are then trained by weight adjustment. Classification involves polling the classifiers and returning the class label that achieves the greatest weighted majority of predictions. Unfortunately, these algorithms have not been well-studied empirically. Blum (1995) produced the only known empirical study using a subset of a calendar scheduling domain (Mitchell et al. 1994), but is more of a study of the incremental learning properties of these algorithms than a study of their ability to track changing concepts.

The FLORA systems (Widmer and Kubat 1996) are a collection of inductive rule learning systems for tracking concept change. FLORA2 uses an adaptive memory-based forgetting scheme to implement a partial memory model, which is discussed further in Section 3.5. FLORA3 extends FLORA2 and is capable of handling changing contexts by storing and retrieving past concept descriptions. An example of a changing context is the seasons. While we may have the concept of warm, this concept changes from season to season. That is, what we consider warm during the summer months is different during the winter months. FLORA4 extends FLORA3 by using mechanisms for coping with noise that are similar to those in IB3 (Aha et al. 1991).

Extensive experiments were conducted using the FLORA systems on the "STAGGER concepts" (Schlimmer and Granger 1986). FLORA2 performs comparably to STAGGER in terms of predictive accuracy and convergence, but is not as sensitive to over-training as STAGGER, because of FLORA's adaptive forgetting policy. As a context reoccurrence experiment, three sets of STAGGER concepts were repeated in order. Comparisons were made between FLORA2 and FLORA3. FLORA3 achieved higher predictive accuracy when contexts reappeared because of its ability to store and retrieve past concept descriptions.

Experiments were also conducted between FLORA2, FLORA3, and FLORA4 with varying amounts of random noise, varying rates of change, varying extents of change, and irrelevant attributes. To summarize, FLORA4 initially converges more slowly than its two predecessors, FLORA2 and FLORA3, on the noise-less STAGGER concepts because of the noise handling mechanisms. Eventually however, FLORA4 achieves the highest predictive accuracy. In the presence of 10, 20, and 40% random noise, FLORA4 once again converges more slowly than the other two, but eventually achieves a higher predictive accuracy.

These learners were also compared with varying rates of drift (slow, medium, and fast). Rates of drift represent how quickly transitions occur from one concept to the next. For example, the slower the rate of drift, the longer the period of uncertainty between concept change. During this period of uncertainty, a given training example could probabilistically belong to either the old concept or the new concept. All three performed comparably on slow moving concepts, with FLORA4 performing the best at higher rates of change.

The FLORA systems were tested for their sensitivity to the extent of concept drift. The extent of concept drift is a measure of dissimilarity between two concepts learned at different times. Again, FLORA4 proved to be the best learner. The final set of experiments involved repeating the previous experiments, but adding irrelevant attributes. These learners did show degradation of performance due to irrelevant attributes. This degradation is a result of FLORA's concept formation process in which conjunctive descriptions are generalized minimally to cover a training example.

Finally, Widmer (1996a) examines how a system can learn and track changing contexts. The main idea is best illustrated with an example. A system may have the concept of warm, but in reality, the concept of warm is dependent on the time of the year, the location, and the like. The seasons are an example of a cyclic context. The learning

systems MetaL(B) and MetaL(IB) are able to discover the attributes that are contextually dependent. Assuming the seasons are the context, then perhaps a contextual attribute is the mean daily temperature. As the context changes, these learners are able to adjust to context change by monitoring the context dependent attributes. Concepts are then either adjusted if the context is new, or simply retrieved and modified if the context is recurrent.

3.5 Aging and Forgetting

Little work has been reported in the machine learning literature on aging and forgetting mechanisms. Aging is a process in which time is used to affect the influence of training examples and concept descriptions in learning or decision processes. Forgetting, on the other hand, is a learning process that explicitly removes training examples or concept descriptions, although others have proposed broader definitions (Markovitch and Scott 1988). This equates to a deductive deletion transmutation, as defined by the Inferential Theory of Learning (Michalski 1994). Forgetting techniques can be memory-based, time-based, proximity-based, frequency-based, or some combination of these. Memory-based forgetting techniques are those that limit the number of examples kept and remove examples when the limit is surpassed. Time-based forgetting involves removing examples that are older than a certain age. Proximity-based techniques delete examples based on their proximity to other examples in the representation space. Finally, frequency-based techniques remove examples based on counting metrics, such as the frequency of occurrence of an example.

Given the restriction of forgetting defined as a removal process, aging and forgetting have not been the subject of many research studies, although in some, forgetting is present, but not focused upon. Freivalds et al. (1995), for example, formally examine the impact of forgetting on learning. Using a formal learning model called Inductive

Inference Machines (IIM), they prove that any function can be learned with a memory model having a linear space complexity (i.e., memory can grow only linearly). If the class of functions is restricted to those computable by finite automata, then a constant amount of memory is sufficient for learning.

More empirical studies have also been conducted. The FAVORIT system (Krizakova and Kubat 1991; Kubat and Krizakova 1992), which uses a decision tree representation, is an extension of EPAM (Feigenbaum 1963). FAVORIT can positively or negatively reinforce training examples with respect to time (i.e., the older an example, the more important, and vice versa). Two types of forgetting are present. The first is a pruning mechanism that is present in UNIMEM (Gennari et al. 1989) in which low-valued nodes are removed. The second is a process in which aging weakens unused knowledge to a point that it contributes little to learning. Experiments were conducted using the Soybean data set (Michalski and Chilausky 1980) with varying amounts of random noise (i.e., 0, 5, 10, and 20%) and measuring how different "intensities" of forgetting affected concept complexity, recognition time, and predictive accuracy. Experimental results show that increased forgetting leads to smaller concept descriptions and lower recognition times, for all levels of noise. Predictive accuracy is unaffected for low intensity forgetting, but as forgetting becomes more intense, predictive accuracy drops to unacceptable levels (< 60%).

The DARLING system (Salganicoff 1993), a memory-based learning system, uses aging and forgetting mechanisms to track changing concepts. Each training example is assigned a weight that is decremented at a rate dependent on the number and proximity of new training examples within an *m*-nearest neighborhood. Experimental results show how the system is able adapt to malfunction (i.e., a finger jam) in a robot grasping domain, but no comparisons are made to the learning system with forgetting disabled.

Sablon and De Raedt (1995) extend Iterative Version Spaces (Sablon et al. 1994), or ITVS, using compacting and forgetting mechanisms to remove redundant instances in

the sets of generalizations and specializations. Theorems are presented showing that extended Iterative Version Spaces preserve the worst case time and space complexities of the original ITVS algorithm. No empirical results are presented.

The FLORA systems (Widmer and Kubat 1996), which are also systems for coping with concept drift, use a memory-based forgetting technique with an adaptive window size. The size of the window is heuristically set based on system performance. If the system is performing well, then the window is decreased to a lower bound. On the other hand, if the system is performing poorly, it is assumed that this poor performance is due to concept change, and the window size is enlarged. Performance is measured using predictive accuracy, the size of the current concept description, and the number of training examples covered by the current concept description. Experimental results for the FLORA systems were discussed in Section 3.4.

Chapter 4: Methodology

4.1 Baseline Progressive Learning

Before discussing the PPML methodology, it is useful to introduce the notion of a baseline progressive learning algorithm. Virtually any batch machine learning algorithm can be made to learn over time (i.e., progressively). We therefore define Algorithm 1 as the Baseline Progressive Learning Algorithm.

Although simple and perhaps obvious, the Baseline Progressive Learner (BPL) has several desirable properties, the most important of which is that the algorithm is, in a few aspects, a worst-case algorithm for a given learner. The algorithm will have the worst-case memory requirements and learning times. On the other hand, if we can assume a perfect data source and learner, the predictive accuracy of this algorithm will be optimal. Unfortunately, we cannot say much about concept complexity in this case since complexity is related more to the distribution of examples in the representation space and the concept representation language rather than the number of examples in the space (Thrun et al. 1991).

Algorithm 1: Baseline Progressive Learner (BPL)
Given data sets $Data_t$, for $t = 1..\infty$
1. $TrainingSet_0 = \varnothing$
2. for $t = 1$ to ∞ do
3. $\quad TrainingSet_t = TrainingSet_{t-1} \cup Data_t$
4. $\quad Concepts_t = Learn(TrainingSet_t)$
5. end

The motivation for introducing the notion of a baseline progressive learner is two-fold. First, for a given learning algorithm (step 4), any progressive version of the algorithm must be an improvement over the BPL in terms of predictive accuracy, memory requirements, learning time, concept complexity, or other measures. Second, experimental comparisons of the progressive version of the learning algorithm can be made to its baseline counterpart. In effect, the BPL is the progressive version of the learning algorithm with its added progressive extensions disabled. This is the comparison approach taken by Aha and Kibler (1987) and Reinke and Michalski (1988).

From an experimental point of view, the baseline learner has an additional important property. It is common in the literature to find comparisons between two different learning systems demonstrating that one rivaled the other in terms of predictive accuracy. Yet to a large extent, such comparisons might be unfair since each learner has representational and inductive biases that could account for the difference in performance. Consequently, it seems that any performance evaluation of a progressive learning system should be made first to its baseline algorithm counterpart. This way, we have, in some sense, controlled for unwanted variability and interactions.

This is of course not to say that comparisons between various learners are not interesting. It ultimately depends on what one is attempting to demonstrate. For the most part, we are empirically examining the effect and or lack of effect of certain mechanisms that appear to be important for learning concepts over time. It therefore seems sensible to make comparisons to the learner without the added mechanisms. For the STAGGER concepts (Schlimmer and Granger 1986), however, comparisons are made to the FLORA systems (Widmer and Kubat 1996). We also compare AQ-PM to AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988) using their original memory models (no and full memory models, respectively) and using a partial memory model.

4.2 Model of Time

Few progressive learning systems have an explicitly stated model of time. Admittedly, a model of time is not as important in systems such as STAGGER (Schlimmer and Granger 1986) and Winnow (Littlestone 1991), since they are predominantly reactive, weight learning algorithms and do not explicitly forget aged examples or concepts. Yet for systems that are expressly designed to deal with temporal phenomena, such details should be considered.

This methodology espouses a linear, discrete model of time. Temporal events are represented using intervals, during which the system can receive zero or more new training examples. This clarification is important because the system can have a richer understanding of time than other systems that restrict themselves to receiving one example per time interval (e.g., STAGGER, Winnow, and FLORA). Because one or more training examples can arrive during an interval of time, we can use the intervals to model arbitrary periods of time, such as days or weeks, without having these periods of time explicitly tied to the arrival of a certain number of new training events. Furthermore, because we allow that no training events might arrive during a time interval, the system is able to track the passage of time.

4.3 Methodology Overview

The main objective of the Progressive Partial Memory Learning Methodology is to learn concepts over time quickly and efficiently. Although one may argue that all progressive learning algorithms learn over time, many of these systems view time simply as a conduit for receiving new training examples, instead of using time to gain potential performance advantages by using mechanisms such as aging, forgetting, and inductive support.

The architecture for PPML, shown in Figure 1, consists of integrated modules for learning and reasoning. Note that learning can be performed by either incremental or temporal batch learning methods. Training examples (i.e., attribute value vectors with class labels) and observations (i.e., attribute value vectors without class labels) flow from the environment to the learning and decision modules (arc 1), and to the teacher (arc 2). The teacher could be a domain expert, the system's user, the system itself, or some combination of these. Since the system operates over time, the teacher may be a different entity during different periods of time. The teacher may or may not give feedback to the learning module about an observation coming from the environment or a decision coming from the decision module (arc 3). Without feedback from the teacher, the system must assume that it is performing properly.

On the other hand, the teacher may choose to classify an observation or provide feedback for a correct or incorrect decision. If the teacher classifies an observation, the system uses its current set of concepts or hypotheses to classify the training example. If the example is classified correctly, then the system may strengthen its beliefs for its current concepts (arc 4), representative examples (arc 5), or both. If the example is not classified correctly, then the system learns from the example and updates the current concepts and representative examples.

The teacher also may decide to provide feedback after the system makes a decision. If the teacher indicates that the system performed correctly, then it may strengthen its beliefs in the concepts that produced the decision or in the representative examples from which the concepts were induced. If the teacher indicates that the system performed incorrectly, then the system behaves as if it receives a newly classified example and modifies its current concepts and representative examples accordingly.

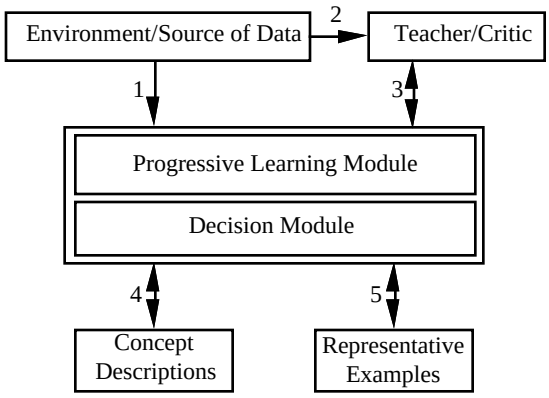


Figure 1: PPML Architecture.

Algorithm 2 is the high-level PPML algorithm that implements the architecture pictured in Figure 1. Hiding within the blocks of the PPML Architecture are various mechanisms and models that manage the selection of the representative examples, the role the current concepts play in induction, if and how representative examples are aged and forgotten, and the like. These ideas are discussed in further detail in the following sections.

Referring to Algorithm 2, the system begins with no concepts and no representative examples, although it may possess an arbitrary amount of background knowledge (Michalski 1983). The variable *t* functions as a temporal counter. No restrictions are placed on the set $Data_t$: its cardinality can be greater than or equal to zero. Note that this is a deviation from traditional thinking in the machine learning community. Many stipulate that the set $Data_t$ must contain only one example and view learning from more than one example in a time step as a type of batch learning. We do not adopt this convention here because such a conception of time is too restrictive.

When the temporal counter *t* is equal to 1, the system will have no concepts or representative examples. Consequently, the first learning step will be from the first data

partition $Data_1$ (steps 4–6). The concepts induced during this learning step will be used to determine which of the examples in the set $TrainingSet_1$ are representative (step 7), using the method described in Section 4.4.

Algorithm 2: Progressive Partial Memory Learning
Given data sets $Data_t$, for $t = 1..\infty$
1. $Concepts_0 = \varnothing$
2. $Representatives_0 = \varnothing$
3. for $t = 1$ to ∞ do
4. $Missed_t = FindMissedExamples(Concepts_{t-1}, Data_t)$
5. $TrainingSet_t = Representatives_{t-1} \cup Missed_t$
6. $Concepts_t = Learn(TrainingSet_t, Concepts_{t-1})$
7. $Representatives_t = FindRepresentativeExamples(Concepts_t, TrainingSet_t)$
8. end

For time steps $t > 2$, the system determines if its concepts require updating by comparing the decisions made by the decision module with those provided by the teacher. If one of the system's decisions does not match the teacher-provided decision, the training example is added to the set of misclassified training examples $Missed_t$ (step 4). The missed examples are unioned with the current set of representative examples, although these sets will be disjoint, to form the new training set (step 5). This new training set and the current set of concepts are then used for learning (step 6).

The new concepts induced by the learning algorithm are used to select representative examples from the training set (step 7). This selection procedure is discussed in detail in Section 4.4. The learning process repeats indefinitely.

4.4 Representative Example Selection

By taking a partial memory approach, we commit to maintaining some of the seen training examples. The key question is, Which examples should be kept? Ultimately, we would like to keep the training examples that best represent a concept. A representative example is defined as a training example that maximally expands or constrains the characteristic

description induced from a set of training examples. Based on the discussion in Section 3.3, the nature of representative examples as they are considered here differs from that discussed by Michalski and Larson (1983) and by Kibler and Aha (1987). The method described here is a post-learning process (cf. Michalski and Larson 1983), and since concept descriptions are used to compute representative examples, then for a given training set presented during a single time interval, there are no ordering effects (cf. Kibler and Aha 1987). PPML is susceptible to ordering effects across time intervals, as are most progressive learning methods, but not within time intervals.

Algorithm 3 computes the set of representative examples from a characteristic concept description and a set of training examples. Specifically, the algorithm returns the training examples that lie on the edges of the hyper-rectangle formed by each decision rule. Variants of this algorithm will compute the sets of representative examples that lie on the corners or surfaces of the hyper-rectangles formed by each decision rule.

Algorithm 3: FindRepresentatives
Given a set of characteristic concept descriptions $Concepts_t$ and a set of training examples $Train_t$ from which $Concepts_t$ were induced.

1. $Representatives_t = \varnothing$
2. removeRanges($Concepts_t$)
3. for each rule $\in$ $Concepts_t$ do
4. for each selector $\in$ rule do
5. newRule = extendRange(selector, rule)
6. $Representatives_t = Representatives_t \cup$ strictMatch(newRule, $Train_t$)
7. end
8. end

While Algorithm 3 easily handles linear, continuous, and cyclic attributes, nominal attributes present a problem because of their unordered nature. If we have a representation space consisting entirely of nominal attributes and a set of characteristic concept descriptions induced from a set of training examples, a set of representative examples can

be easily found. The problem that arises as a result of the unordered nature of nominal domains is there exists some projection of the representation space for which any removed training example is representative. To handle this situation, selectors containing nominal attributes are skipped by the removeRanges (step 2) and the extendRange (step 5) functions.

The function removeRanges (step 2) simply removes intermediate attribute values from each selector of the concept description. Given the characteristic concept description [x1 = 1 ∨ 2..5 ∨ 7] & [x2 = 1 ∨ 3 ∨ 5], removeRanges produces [x1 = 1 ∨ 7] & [x2 = 1 ∨ 5].

The function extendRange (step 5) extends the interval for the given selector in the rule. For instance, given the first selector of the rule [x1 = 1 ∨ 7] & [x2 = 1 ∨ 5], extendRange produces [x1 = 1..7] & [x2 = 1 ∨ 5].

At this point, we have created syntactically a rule that will match the training examples that lie on the hyper-rectangle edge produced by the current selector, in this case [x1 = 1..7]. The function strictMatch (step 6) simply returns those training examples that match the syntactically modified rule.

Figure 2 is a diagrammatic visualization of the setosa class and a portion of the versicolor class from a the Iris data (Fisher 1936) taken from the University of California, Irvine machine learning repository (Merz and Murphy 1996). The original data set was discretized using the SCALE implementation (Bloedorn et al. 1993) of the ChiMerge algorithm (Kerber 1992). Each example of an iris is expressed using four attributes: petal length (pl), petal width (pw), sepal length (sl), and sepal width (sw). The characteristic concept descriptions induced from these training examples are:

```
    setosa  <::  [sl = 0..3] & [sw = 0 ∨ 2..4] &
                 [pl = 0] & [pw = 0]

versicolor  <::  [sl = 1..6] & [pl = 1] & [pw = 1..2]
```

These concepts are visualized in Figure 3. Note that the training examples are overlaid on the concept descriptions. After the FindRepresentatives algorithm is applied to the concept descriptions and training examples in Figure 3, those training examples that lie on the borders of the concept descriptions are kept as representative examples. Figure 4 shows the corresponding representative examples computed for the two classes. Note that rules are matched to training examples in *n*-dimensional space, which is why the following example is judged representative:

```
versicolor  <::  [sl = 3] & [sw = 0] &
                 [pl = 1] & [pw = 2]
```

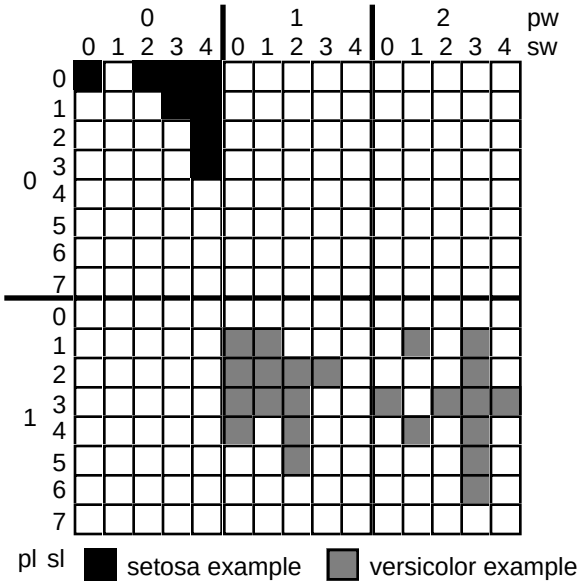
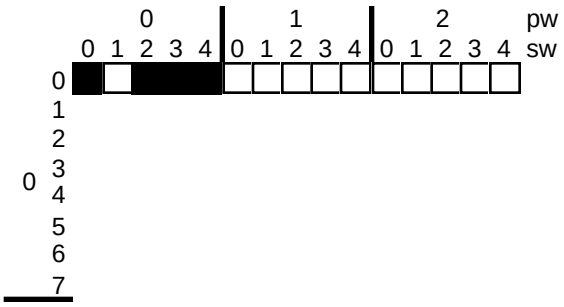


Figure 2: Visualization of the setosa and versicolor training examples.


0
1
2
pw
0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 sw
0
1
2
3
0 4
5
6
7
1
pl sl

Although the characteristic concept description is used for judging representative examples, other types of descriptions (e.g., discriminant descriptions) can be used for reasoning. The operation of finding representative examples is a deductive selection knowledge transmutation, as defined by the Inferential Theory of Learning (Michalski 1994).

In general, using this method, the number of representative examples retained will be as follows. Assume $D_i <:: C_j$ is a characteristic decision rule induced from the training set $Train_t$ for some time step t over the n-dimensional discrete representation space $\Re$. By definition, the following are equivalent:

1. The dimensionality n of the representation space $\Re$.

2. The number of selectors $s_k \in C_j$, and

3. The number of attributes forming the representation space $\Re$.

For the baseline progressive learner, each attribute value of a training example in $Train_t$ will map to a corresponding referent value in a selector $s_k \in C_j$. For a set of training examples $Train_t$, each attribute will result in a selector $s_k \in C_j$ such that the size of the selector's referent is equivalent to the number of the attribute's values. Therefore, the number of training examples, i.e., $|Train_t|$, will be equal to:

$$\prod_{k=1}^{n} |referent_k| \tag{1}$$

For the partial memory learner, each selector $s_k \in C_j$ will match with at most $|referent|$ training examples, since each referent value in a selector maps to one training example. Further, each selector $s_k \in C_j$ corresponds to 2^{n-1} edges of a hyper-rectangle realized by the complex C_j. Therefore, the number of training examples matched by one selector $s_k \in C_j$ is

$$2^{n-1}|referent_k|$$

If we were to compute this number for each selector s_k in the complex C_j, for $k = 1..|C_j|$, then we would over-count the training examples that lie at the corners of the hyper-rectangle. Therefore, we should subtract the two training examples that lie at the endpoints of each edge of the hyper-rectangle:

$$2^{n-1}(|referent_k| - 2)$$

Now we can compute this term for each selector in the complex C_j (i.e., each dimension of the hyper-rectangle):

$$\sum_{k=1}^{n} 2^{n-1}(|referent_k| - 2)$$

But this under-counts the number of training examples matched because we excluded the corners. Since there are 2^n corners in an n-dimensional hyper-rectangle, the total number of examples matched for an n-dimensional representation space $\Re$ is:

$$\sum_{k=1}^{n}\left[2^{n-1}(|referent_k| - 2)\right] + 2^n, \text{ for } |referent_k| \geq 2 \qquad (2)$$

As one can see by examining formulae 1 and 2, for the worst case, the partial memory learner will retain fewer training examples (by a multiplicative factor of |*referent*|) than the temporal batch learner.

As an example, refer to Figure 2, which is a visualization of the Iris data set (Fisher 1936). The setosa class consists of 8 examples, while the versicolor class consists of 23. The characteristic concept description for the setosa class is:

```
setosa      <::   [sl = 0..3] & [sw = 0 ∨ 2..4] &
                  [pl = 0] & [pw = 0]
```

The size of the referents (sl, sw, pl, and pw) for the setosa characteristic description is 4, 5, 1, and 1. From formula 1, the worst case size of the training set is 20.

For the partial memory learner, the size of referents greater than 2 are sl and sw and have a domain size of 4 and 5, respectively. Therefore, $n = 2$. From formula 2, the worst case size of the number of training examples judged representative is 14.

For the versicolor class, the characteristic concept description is:

```
versicolor   <::   [sl = 1..6] & [pl = 1] & [pw = 1..2]
```

The selector for the setosa width (sw) is not shown in the characteristic description because all values for the selector are present. The size of the referents (sl, sw, pl, and pw) is 6, 5, 1, and 2. The worst case number of examples for this class is 60, computed by formula 1.

For the partial memory learner, the referents greater than 1 are the referents for the sl, sw, and pw attributes. Therefore, $n = 3$. From formula 2, the worst case number of training examples judged representative is 36.

4.5 Aging Mechanisms

In the progressive partial memory learning methodology, aging mechanisms are those procedures and policies that determine how the ages of representative and training examples influence the selection of concept descriptions. If the training example is judged to be representative, then the example and its age will be used in subsequent learning steps (given appropriate parameter settings). Various policies exist for assigning or updating the age of a representative example, which is necessary since representative examples may be re-evaluated at each learning step. In addition, new training examples may arrive to the system that are duplicates of existing representative examples. One policy is to keep a representative example's initial age. Another is to re-assign the representative example's age if a new training event arrives that matches the representative example. Finally, the representative example's age may be updated each time it is re-evaluated and deemed yet again representative.

The ages of representative and training examples influences rule selection by an evaluation of criterion from a lexicographic evaluation function, or LEF (Michalski 1983). This function is a set of criterion-tolerance pairs and is used to select one of many valid inductive assertions generated by a learner. A set of valid inductive assertions is reduced by evaluating the assertions according to a criterion and keeping those that are within a specified tolerance. This process continues until the set of assertions is a singleton, or until the set of criterion-tolerance pairs has been exhausted, in which case all assertions are equally preferable with respect to the LEF.

The aging criterion for the LEF are the sums of functions that take an example's age as their argument and monotonically increase or decrease into the past (Maloof and Michalski 1995c). Monotonically increasing functions with respect to the past are useful for selecting concept descriptions that are induced from older examples rather than newer ones. Conversely, monotonically decreasing functions with respect to the past are useful for selecting concept descriptions that are induced from newer examples rather than older ones. That is, the cost c_j for the complex C_j is

$$c_j = \sum_{i=1}^{n} f(age(e_i)) \tag{3}$$

where

n is the number of representative and training examples covered by complex C_j,

e_i is the ith training example,

$age(x)$ is a relation that returns the age of a training example, and

$f(x)$ is a monotonic function.

$f(x)$, for instance, can be a constant function in which all examples contribute equally to a concept description regardless of their age. On the other hand, it could be $\exp(-x)$, in which case, there will be a strong preference toward concept descriptions that cover old training examples. Alternatively, the aging function $f(x)$ could be learned.

4.6 Inductive Support Mechanisms

In the methodology of progressive partial memory learning, inductive support mechanisms are those methods that augment representative examples with frequency counts and the rule selection methods that use these counts. In atemporal batch learning, the opportunity or perhaps the need for inductive support mechanisms is not as great as it is when learning over time. Over time, the system may make repeated observations. Perhaps the email from the Dean is always read first. What types of weights and mechanisms are appropriate and can be leveraged to enhance rule learning?

Although many conceptions of probability for inductive support exist (Cohen 1989; Schum 1994), a Pascalian conception is appropriate for many machine learning contexts, especially those that involve static representation spaces. Such a conception is appropriate for situations in which one can apply counting arguments to derive the probability that a certain event may occur. This follows from the assumption of a static representation space. Consequently, the PPML methodology uses a simple frequentistic count to augment representative examples.

The number of training examples covered by a rule is often used as a metric of rule strength. Another dimension is how often the system has seen the various training examples. The system can measure rule strength not only as a measure of how many training examples the rules cover, but also as a measure of how often those covered training examples have been seen. Mechanisms exist, much like those for aging, that select rules based on their coverage of frequently or infrequently seen examples. These rule selection criteria are implemented using the LEF and can be used alone or in conjunction with other rule selection criteria, such as those for aging.

4.7 Forgetting Mechanisms

Three primary types of forgetting mechanisms are espoused by the PPML Methodology: memory-based, time-based, and frequency-based. Memory-based forgetting mechanisms keep a specified number of representative examples in memory. A secondary forgetting policy is used to remove undesirable examples as new ones are remembered. For example, the system may always forget the oldest examples. Alternatively, it may forget examples covered by hypotheses of low weight (i.e., hypotheses that cover few examples).

Time-based forgetting policies use the age of an example as the main determinant for removal. Although time-based forgetting is straightforward since examples older than a certain time are removed, such policies become more complex if the system adapts this threshold. As discussed previously, Widmer and Kubat (1996) take this approach.

Frequency-based approaches use some measure of the example's weight to determine whether it is kept or removed. Assume an example's weight is simply a count of the number of times the system has observed the example. Therefore, representative examples of low weight may be candidates for removal. The inherent problem with frequency-based schemes is that some mechanism must keep new examples for a period of time so they have the opportunity to achieve a level of certainty. In other words, if our policy is to forget examples of low weight, then all new examples will be forgotten in the next learning step since they invariably will be of low weight. Therefore, some time-based policy must be enacted to ensure that new examples have an opportunity to accumulate weight. Again, the frequentistic threshold at which examples are forgotten or the duration for which examples are kept before they are candidates for removal could be determined adaptively.

Chapter 5: AQ-PM System Description

To validate the PPML Methodology, selected components of the methodology were implemented in an experimental system, called AQ-PM, which is an extension of the inductive learning system AQ15c (Wnek et al. 1995). The elements of the PPML Methodology that were integrated into AQ15c are accessed by setting various learning parameters. The following sections provide a brief description of the AQ-PM learning system.

5.1 Invoking Partial Memory Learning

The most important parameter is the one that invokes partial memory learning. This is accomplished by setting the `learning` parameter to `partial`. The other valid learning parameter is `batch`, the default, which invokes AQ15c batch learning (see Wnek et al. 1995).

Inductive learning (i.e., the Learn function on line 6 of Algorithm 2) is carried out by determining the cover $C(e_1 / e_0)$ of a set e_1 against e_0 using the AQ algorithm (Michalski 1969). A training set is formed by the union of the representative examples and any new, misclassified training examples. Specifically, a positive concept description is induced by extending the positive training set against the negative set. That is,

$$Concepts_t^+ = C(TrainingSet_t^+ / TrainingSet_t^-)$$

Negative concept descriptions are computed in a similar manner. In a multiple class learning situation, a cover is computed for each class with the set of negative training examples consisting of the training examples from classes other than the current one.

5.2 Selecting Representative Examples

Once partial memory learning has been selected using the `learning` parameter, two schemes are available for representative example selection. This is set using the `intersect` parameter. Presently, the valid settings are `borders`, which is the default, and `corners`. The `borders` selection method behaves as described in Section 4.4. This method returns the training examples that lie on the borders of the hyper-rectangle expressed by a set of characteristic concept descriptions.

The second setting, `corners`, is a simplification of the borders method and returns those training examples that lie on the corners of the hyper-rectangle expressed by a set of characteristic concept descriptions. An alternative method, an extension of the borders method, is to return the training examples that lie on the surfaces of the hyper-rectangle expressed by a set of characteristic concept descriptions. The surfaces method was not implemented.

The AQ-PM system, which is an extension of the AQ15c system (Wnek et al. 1996) uses an attributional language. Continuous attributes are handled by abstracting attribute values into a discrete, linear domain. Linear and cyclic attributes are handled easily by the representative example selection mechanisms. Nominal attribute values, including structured attributes, are difficult to handle from a representative example selection standpoint, but not an inductive learning standpoint. These difficulties were discussed in Section 4.4. Consequently, selectors involving nominal or structured attributes are matched directly to training examples without modifying the selector's referent.

5.3 Updating Representative Examples

Once representative examples have been selected, there are various ways these new representative examples can be added to the current set of representative examples. Two

methods have been implemented using the `update` parameter: `reeval`, the default, and `accum`. The `reeval update` parameter re-evaluates the old representative examples and any new, misclassified training examples. Therefore, at each time step, each representative example is evaluated and is a candidate for removal. This method appears in the original progressive partial memory learning algorithm (Algorithm 2).

The `accum update` parameter accumulates new representative examples into the set of current representative examples. With this method, once an example becomes a representative example, it will remain so, unless removed by some explicit forgetting process. Algorithmically, instead of computing the new set of representative examples from the current set and any missed examples, as is the case with the `reeval update` parameter, with the `accum update` parameter, we compute the new portion of the representative examples from those examples that were originally missed and add these to the current set of representative examples. This is expressed below in Algorithm 4. Note that Algorithm 2 and Algorithm 4 differ only on line 7.

Algorithm 4: PPML (Accum updating)
Given data sets $Data_t$, for $t = 1..\infty$
1. $Concepts_0 = \varnothing$
2. $Representatives_0 = \varnothing$
3. for $t = 1$ to ∞ do
4. $Missed_t = FindMissedExamples(Concepts_{t-1}, Data_t)$
5. $TrainingSet_t = Representatives_{t-1} \cup Missed_t$
6. $Concepts_t = Learn(TrainingSet_t, Concepts_{t-1})$
7. $Representatives_t = Representatives_{t-1} \cup$ $FindRepresentativeExamples(Concepts_t, Missed_t)$
8. end

5.4 Using Input Hypotheses as Events

In the default mode, AQ-PM converts input hypotheses into training events and learns from these events, representative examples, and misclassified training examples. This takes advantage of a procedure present in the AQ15c learning system (Wnek et al. 1995) for

optimizing input hypotheses. AQ-PM can also learn without converting input hypotheses into events. This function is set in AQ-PM using the `useinhypos` parameter, which takes the values `yes`, the default, or `no`.

5.5 Ambiguous Examples

An ambiguous example is one that belongs to more than one class. Although the AQ systems have the `ambig` parameter for dealing with ambiguous examples, learning over time introduces additional problems that are not present in atemporal batch learning. For example, suppose we have a two class learning problem in which concepts are changing over time. Depending on the rate of change and the size of the representation space, it might be likely that an example that was positive at time t_i is now negative at time t_j, for $i < j$. Suppose further that the example is a representative example in the positive class and a new training example in the negative class. Which example should be used for learning?

Ultimately the answer to this question depends on the problem being addressed. Consequently, the system should allow the designer to configure the learning system to handle the situation properly. This illustration, however, outlines the need for a time-based `ambig` parameter. The setting for this new `ambig` parameter is `recent`. If one is learning changing concepts and assumptions cannot be made about noisy examples or the speed at which concepts change, then a reasonable heuristic is to focus on the most recent training examples. This heuristic is observed by most, if not all, progressive learning systems that cope with changing concepts. The `recent ambig` parameter, therefore, simply gives preference to the example that is newest. Referring to the previous illustration, the new negative training example would be used for learning, and the older positive representative example would be discarded for that learning step (i.e., not forgotten).

5.6 What Time is it?

It has been stressed that the PPML Methodology, and hence AQ-PM, has an explicit model of time. Since AQ-PM is invoked at each time interval, the current time must appear as a parameter. Time is an integer in the range [0, 150]. The limit of 150 is arbitrary and can be extended to the maximum integer allowed on any given machine. The number 0 therefore marks the beginning of time. This parameter is used by the aging and forgetting mechanisms.

Time is specified using the `time` parameter. Usually, the `time` parameter is not set directly by the user. AQ-PM initializes the `time` parameter to 0 when it receives an input file that specifies partial memory learning, has only training examples, no representative examples, and no input hypotheses. After learning, the `time` parameter is incremented and output. These output parameter settings are intended to be used as input parameter settings for the next learning step.

5.7 Forgetting

Forgetting involves removing representative examples that meet some user-specified criterion. This function is implemented using the `forget` parameter. Presently, its settings are `off`, which is the default, and `t`<*n*>, where *n* is some integer. If the `forget` parameter is set to `off`, then intuitively, no forgetting mechanisms are active.

The `t`-parameter specifies a time-based forgetting function in which representative examples older than *n* time steps are forgotten. This is implemented by simply removing an example if its age is less than the current system time minus *n*. Memory-based and frequency-based forgetting functions have not been implemented.

5.8 Inductive Support

Inductive support mechanisms are similar to aging mechanisms. While aging mechanisms are time-based, inductive support mechanisms are frequency-based. The implemented functions for inductive support deal with rule selection using the LEF. Frequency-based flexible matching schemes were not implemented.

Using the criterion table, which implements the LEF in AQ15c (Wnek et al. 1995), two methods were implemented for taking advantage of the frequency of an event's occurrence. The `maxseen` setting gives preference to rules that cover the most frequently seen examples. Conversely, the `minseen` setting gives preference to rules that cover the least frequently seen examples.

The frequency of an example is updated when new training examples are processed. For each representative example that is present in the set of new training examples, a counter is incremented that indicates the number of times that representative example has been observed since the beginning of time.

5.9 Aging

Aging involves how the age of an example affects matching and rule selection. Four aging functions have been implemented: constant, linear, exponential, and logarithmic. These are specified using the `aging` parameter in conjunction with the criterion table. Time-based flexible matching routines were not implemented.

The `aging` parameter takes three values: `off`, the default, `exp`, and `log`. The `off` setting disables nonlinear aging, `exp` uses an exponential aging function, and `log` invokes a logarithmic aging function. The new criterion table settings can be either `maxold`, in which rules covering the most old examples are preferred, and `minold`, in which rules covering the fewest old examples are preferred. The aging policy is set using a combination of the `aging` parameter and the time-based criterion settings. For example, to set a constant or

uniform aging policy in which all examples are treated the same regardless of their age, then the `aging` parameter would be set to `off`, and no criterion table would be used. This is the default. On the other hand, to specify a policy in which rules covering older examples are preferred and these examples are aged linearly, then the `aging` parameter would be set to `off`, and the criterion table would be set to `maxold`. Finally, to specify a policy that prefers rules covering new examples and the age of the examples are weighted exponentially, then the `aging` parameter would be set to `exp` and the criterion table would be set to `minold`.

Presently, an example's age (i.e., the time at which it arrives) is set when the example first arrives to the system and remains this age as long as the example is kept as a representative example. New training examples arriving to the system are assigned a creation time that is equal to the system's current time. An example's age is therefore computed by subtracting an example's creation time from the current system time. Representative examples that are re-evaluated or match new training examples keep their original ages. An interesting area of investigation might be to see if various policies for setting an example's age yields some change in performance and in what situations this occurs.

5.10 Weighting Representative Examples

Applications may arise in which it is necessary to give more or less strength to representative examples versus the newly acquired training examples. While aging weights representative examples in the temporal case, and inductive support mechanisms weight representative examples in the frequentistic case, AQ-PM also provides an atemporal weighting mechanism for representative examples. This function is accessed using the input parameter `repfactor` and the criterion table entries `maxrep` and `minrep`.

The input parameter `repfactor` must be an integer and is an additive weight factor associated with each representative example. The default value for `repfactor` is 1. That is, representative examples and new training examples are of the same weight. If the `maxrep` criterion table entry is used and `repfactor` is equal to 1, then using the LEF, the rule induction process will prefer rules that cover the maximum number of representative examples. If the `repfactor` is 2, then representative examples count twice as much as training examples, and so on. Now, if the `maxrep` criterion is used, the LEF will select rules that have a representative/training example coverage ratio of 1:2. The `minrep` criterion table entry functions similarly, except that a rule's coverage of the representative examples is minimized.

Chapter 6: Experimental Results

To validate the Progressive Partial Memory Learning Methodology, the AQ-PM inductive learning system was experimentally studied using several problems. The first problem is from the computer intrusion detection domain in which a dynamic knowledge-base system must detect use by illegitimate users and misuse by legitimate users of a computer system by comparing a user's short-term behavior to historical behavior. The second is an email sorting problem in which an intelligent agent must sort a user's email into categories, such as important and unimportant. The third is an object recognition problem in which a system acquires a concept description of a visual object that is present in a set of images. The fourth problem is synthetic and involves a changing concept. The real-world problems were selected because they require progressive learning of a changing concept and are relevant to current scientific and societal issues. The synthetic problem was selected because of its use by other researchers investigating methods for learning changing concepts. To place AQ-PM in a context with other progressive learners, comparisons are made to AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988) for the blasting cap detection problem and the computer intrusion detection problem. Finally, the AQ11 and GEM incremental learning algorithms are used as the learning function in the PPML Algorithm (Algorithm 2) for the blasting cap detection problem and the computer intrusion detection problem.

6.1 Experimental Design and Analysis

For the data sets used for experimentation, each was partitioned randomly over time. The number of partitions depended on the size of the data set. A partition is analogous to a time step in Algorithm 2. Furthermore, each partition represents a percentage of the original training data. Consequently, if we speak of learning on the second partition of a 10 partition experiment, then the system is learning from 20% of the data. For each partition, a percentage of the original data was selected randomly and used for training. Testing was then conducted using the original data set. For most problems, two learning systems were compared: AQ-PM and the baseline progressive learner using AQ15c. We can view the baseline progressive learner using AQ15c as AQ-PM with all partial memory mechanisms inactive. Henceforth, we will refer to the baseline learning algorithm (Algorithm 1) using AQ15c as AQ-BPL to distinguish it from AQ15c running in other modes, such as batch mode.

Parameters common to AQ-PM and AQ-BPL were set and held constant for all experiments. Initial experiments were conducted over most, if not all, parameter settings that were particular to AQ-PM to find a near-optimal set of parameters for the given problem. The parameter settings for each of the experiments are shown in Appendices A–D.

For each learning and recognition run, both learners were given the same training and testing data, which is a single-factor, repeated-measures experimental design (Keppel et al. 1992). Several measures were accumulated and computed during the learning runs, depending on the problem. Experimental results are presented for each partition, or time step. For each partition, a random percentage of the original data was randomly selected and used for learning. Testing was then conducted using the original data. This procedure was repeated 100 times for each partition. Each measure presented for a partition is therefore the average of that measure over the number of learning runs dictated by the

validation methodology (i.e., 100 iterations). As an example, if for a partition of the data set (i.e., a time step), a validation methodology of 100 learning and testing runs were used, then the reported predictive accuracy for that partition is the average of the predictive accuracy from each of the 100 iterations.

Comparisons of recognition times for the evaluated methods were not notable and are therefore not reported. Recognition times are virtually identical because both methods use the same matching algorithms on the same testing data. Any difference in recognition time is therefore attributable to concept complexity. Although the concept complexity did differ between the two methods, these differences were not enough to significantly affect recognition rates for the amounts of testing data used.

The best experimental results are presented below, although the results of other experiments are discussed in Section 7. An analysis of variance (Keppel et al. 1992) was conducted for each of the best experiments to determine if the predictive accuracy results are statistically significant and at what level. The particulars of each experimental design and analysis are discussed in greater detail below. Note that statistical significance was not achieved for the first partition of any experiment. The reason for this is simple. For the first partition, AQ-BPL and AQ-PM behave identically. Consequently, there is no, or very little, difference in their performance, statistically or otherwise.

In mapping the experimental design and analysis of a single factor, repeated-measures experiment (Keppel et al. 1992) to a machine learning context, a slight adjustment was made. The statistical analysis is designed to examine the interaction between treatments among subjects. Intuitively, we would assume that the machine learning algorithms would correspond to subjects and each trial of the experiment would correspond to treatments. The statistical analysis is designed to differentiate between treatments of an experiment, and we wish to differentiate between learning algorithms on a specific data set.

Consequently, it is necessary to view each learning algorithm as a treatment and each trial of the experiment as a subject.

6.2 Computer Intrusion Detection Problem

Computer intrusion detection is the detection of computing system use by illegitimate users and misuse by legitimate users (Denning 1988). Typically this is accomplished by comparing a user's short term behavior to a historical profile. Detecting anomalous user behavior is an ideal domain for machine learning because user profiles are formed from low-level computing resource metrics, such as CPU usage. It would be difficult, if not impossible, for anyone to form profiles for a group of users using such low-level measures. Other experiments in this domain have been reported by Maloof and Michalski (1995a, b).

6.2.1 Problem Description

To study this problem, three weeks of computing activity was gathered using the Unix acctcom command (Frisch 1991), which resulted in over 11,200 audit records. The numeric metrics of an audit trail form a multivariate time series. Davis (1981) characterized a time series for symbolic machine learning by computing the average, minimum, and maximum values of a series over a window. A *session* is analogous to a window and is defined as a period of continuous computing activity bounded by either a logout or twenty or more minutes of idle time.

Nine of the computing system's most active users were selected for the study. The audit trail was parsed into sessions for each user and the average, minimum, and maximum were computed for the metrics real time, CPU time, user time, characters transferred,

blocks read and written, CPU factor, and hog factor.[1] This resulted in 21 real or integer attributes. The SCALE (Bloedorn et al. 1993) implementation of the ChiMerge algorithm (Kerber 1982) was used to abstract the representation space. The PROMISE method (Baim 1988) was used to select the 13 most relevant attributes (i.e., those attributes achieving a score greater than 0.9). The representation space for this problem consisted of nine classes, corresponding to the nine users, 238 examples, and 13 linear attributes with levels ranging from 5 to 76. The size of the representation space was 6.7×10^{16} with the training examples representing 3.6×10^{-13}% of the space. The 13 linear attributes forming the final representation space were average and maximum real time, average and maximum system time, average and maximum user time, average and maximum characters transferred, average blocks transferred, average and maximum CPU factor, and average and maximum hog factor. For the experiments presented, the training set was partitioned into 10 partitions. For each partition, 100 learning and testing runs were conducted, measuring predictive accuracy, learning time, concept complexity, and the number of examples maintained. Appendix A shows the learning parameters for this problem.

6.2.2 Experimental Results

The numerical results for the computer intrusion detection experiments are presented in Table 2. The table shows for each partition and learning method the average predictive accuracy with a 95% confidence interval, learning time in seconds, the number of examples maintained, and the concept complexity in both rules and selectors. The predictive accuracy results are statistically significant at $p < .01$. These numeric results are summarized graphically in Figures 5–8.

---

[1]Previous experimentation included the standard deviation of these measures, which produced slight increases in predictive accuracy and significant increases in learning time. Consequently, these attributes were not computed for this set of experiments.

Table 2: Experimental results for the computer security problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)*	Learning Times (sec)	Examples Maintained	Concept Complexity (rules/selectors)
1	AQ-PM	75.68±1.14	14.97	27.0	9.06/10.55
	AQ-BPL	75.68±1.14	14.85	27.0	9.06/10.55
2	AQ-PM	83.30±1.07	18.55	34.61	9.77/14.61
	AQ-BPL	86.02±0.79	21.39	54.0	10.31/15.47
3	AQ-PM	86.45±0.94	21.55	40.89	10.52/17.75
	AQ-BPL	91.26±0.51	27.64	81.0	11.49/19.18
4	AQ-PM	87.92±0.79	24.51	45.95	11.36/20.53
	AQ-BPL	93.65±0.32	32.47	106.0	12.38/22.31
5	AQ-PM	90.20±0.61	27.04	50.87	12.16/23.14
	AQ-BPL	95.56±0.29	37.28	131.0	13.2/25.01
6	AQ-PM	91.51±0.68	29.68	55.03	13.0/25.58
	AQ-BPL	96.82±0.27	41.63	154.0	13.95/26.98
7	AQ-PM	93.12±0.55	31.46	58.49	13.62/27.47
	AQ-BPL	98.01±0.16	45.47	176.0	14.66/28.59
8	AQ-PM	93.92±0.56	33.37	61.53	14.18/29.48
	AQ-BPL	98.78±0.12	48.89	197.0	15.41/29.86
9	AQ-PM	94.73±0.56	35.37	64.28	14.72/30.84
	AQ-BPL	99.44±0.09	52.79	218.0	15.99/30.94
10	AQ-PM	96.02±0.43	36.67	66.85	15.08/32.47
	AQ-BPL	100.00±0.00	55.64	238.0	16.62/31.94

*$p < .01$

Figure 5 shows how the predictive accuracy increases with increasing amounts of training data for AQ-BPL and AQ-PM for the computer intrusion detection problem. As expected, the predictive accuracies for both learners increase as they process more training data. The predictive accuracy performance of AQ-PM is not as high as AQ-BPL, which is to be expected, since AQ-PM uses fewer training examples at each learning step. When learning stops at partition 10, however, AQ-PM is within 4% of AQ-BPL's performance (96.02% vs. 100%).

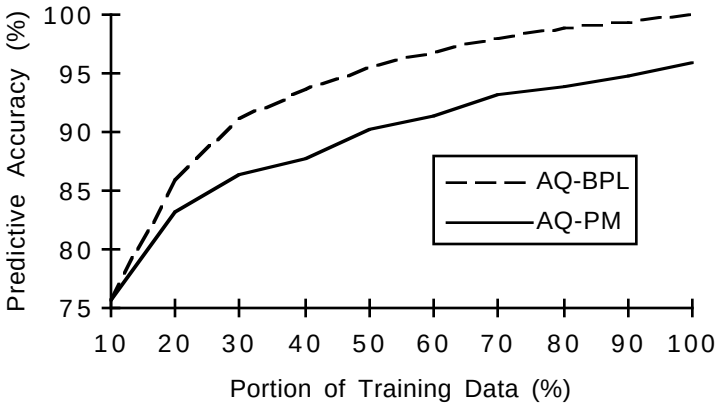


Figure 5: Predictive accuracy for the intrusion detection problem.

Although AQ-PM's predictive accuracy was slightly less than AQ-BPL's, the difference in learning time was greater, as shown in Figure 6. At the end of the experiment, the learning time for partition 10 for AQ-BPL was 52% slower than AQ-PM (55.64s vs. 36.67s). A learning speedup is expected, since AQ-PM learns concepts from fewer examples than the baseline progressive learner AQ-BPL.

A more impressive performance gain can be seen in the number of examples maintained, as shown in Figure 7. For partition 10, AQ-BPL maintained 256% more training examples than AQ-PM (238 vs. 66.85).

Finally, Figure 8 shows the comparison between the two learning methods for concept complexity using selectors. For this problem, the concept complexities for the learning methods were similar for both the number of rules and number of selectors.

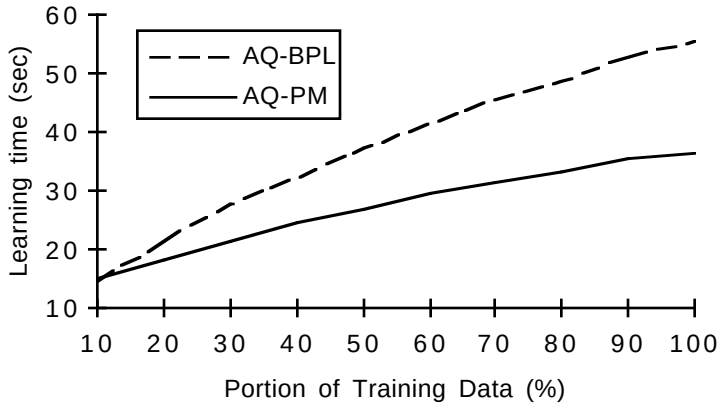


Figure 6: Learning times for the intrusion detection problem.

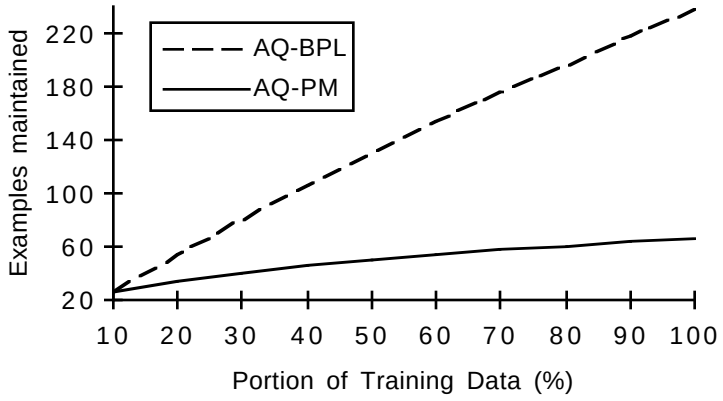


Figure 7: Memory requirements for the intrusion detection problem.

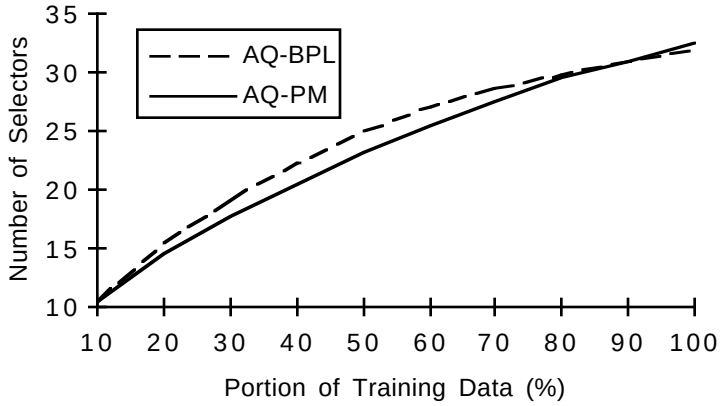


Figure 8: Concept complexity for the intrusion detection problem.

Figure 9 shows examples of rules induced by AQ-PM for two computer users: daffy and coyote. Note that the rules have been written to show the original real ranges. The rule for daffy consists of one selector (or condition) using the average system time attribute. The rule is accompanied by two weights that indicate the strength of the rule. The t-weight, which is 10, indicates how many total representative and training examples are covered by the rule. The u-weight, which is also 10, indicates how many of the total examples covered are covered uniquely by this rule. Rules can potentially overlap; therefore, for a given class, one example may be covered by more than one rule. Daffy's computing behavior is characterized by intense use of the system. This is easy to infer from the single selector rule consisting of a relatively high range for the `averageSystemTime` attribute.

The concept description for coyote is more complex than daffy's. It consists of two decision rules, each consisting of two selectors. Coyote's computing behavior is quite different than daffy's in that it is characterized by a comparative lack of computer use. Coyote's `averageSystemTime` is much less than daffy's, while other measures, such as `averageCharactersTransferred` and `averageBlocks`, also consist of low ranges.

Referring to Table 2, the average number of rules and selectors used to represent concepts for partition 10 is roughly 16 rules and 32 selectors, but this is for 9 classes. The averages per class are about 2 rules and 3 selectors. Therefore, daffy's rule is an example of simple rule for this domain, while coyote's rules are examples of complex rules.

To summarize, for the computer intrusion detection problem, AQ-PM achieved significantly faster learning times (AQ-BPL was 52% slower) and used considerably less memory (AQ-BPL required 256% more memory) with slightly decreased predictive accuracy (4% less). The concept complexity for both methods was comparable.

```
 daffy  <::  [averageSystemTime = 25352.53..63914.66]
             (t-weight: 10, u-weight: 10)

coyote  <::  [averageSystemTime = 3676.80..4579.19] &
             [averageCharactersTransferred = 0.0..0.20]
             (t-weight: 7, u-weight: 6)
coyote  <::  [averageRealTime = 44.75..404.84] &
             [averageBlocks = 0.0..0.39]
             (t-weight: 4, u-weight: 3)
```

Figure 9: Examples of AQ-PM rules for daffy and coyote's computing behavior.

6.3 Email Learning Agent

Intelligent agents are personalized software systems that assist users (Maes 1994). They are knowledge-based systems, but differ from traditional systems in that intelligent agents possess a model of their user's preferences instead of a model of a domain expert. These models can be either pre-programmed or acquired through learning. Several agents have been constructed for a variety of tasks, such as email sorting (Maes 1994), news selection (Bloedorn et al. 1996), and meeting scheduling (Chen et al. 1996). The task considered here is an email sorting task using the email learning agent (ELA) data set (Bloedorn and Michalski 1996).

6.3.1 Problem Description

For this study, three of the original four classes were chosen for one user. One of the classes had too few examples for learning over time, so it was eliminated. The three classes used were immediate attention (IA), read sometime (RS), and unimportant (UI). The domain consisted of eight attributes including the sender's name, the subject, the length of the email, whether it was a reply to a previous email, and so on.

After abstracting the representation space using the SCALE implementation (Bloedorn et al. 1993) of the ChiMerge algorithm (Kerber 1982), the number of levels for

the attributes ranged between 2 and 142. The training set consisted of 223 examples with most of the examples (88%) classified as needing immediate attention (IA). The remaining 12% of the training data was split evenly between the RS and UI classes. The size of the representation space was 1.4×10^{11} with the training examples representing 1.6×10^{-7}% of the space. For the experiments presented, the data was partitioned into 5 partitions or time intervals. For each partition, 100 learning and testing runs were conducted. Measures computed were predictive accuracy, learning time, concept complexity, and the number of examples maintained. The parameters used for this problem appear in Appendix B.

6.3.2 Experimental Results

Table 3 summarizes the numerical results from experiments with the ELA data set. AQ-BPL and AQ-PM were compared using average predictive accuracy, learning time, examples maintained, and concept complexity. Note that an analysis of variance showed that the predictive accuracy results are statistically significant at $p < .01$.

With regards to predictive accuracy (Figure 10), both learning systems achieved high predictive accuracy rates. Looking at the results from the last learning step, Partition 5, the final predictive accuracies are within 4% of each other: 95.98% for AQ-PM and 99.01% for AQ-BPL.

Referring to Figure 11, the learning time comparison for the email learning agent, AQ-PM demonstrated a lower growth rate with respect to learning time than did AQ-BPL. The learning times for AQ-BPL were 66% longer than those for AQ-PM (10.42s vs. 6.29s). Again, this is due primarily to the fact that at each learning step, AQ-PM is learning from fewer examples.

Table 3: Experimental results for the email learning agent.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)*	Learning Times (sec)	Examples Maintained	Concept Complexity (rules/selectors)
1	AQ-PM	90.20±1.01	2.72	45.22	5.41/7.39
	AQ-BPL	90.75±0.92	2.57	46.0	5.32/7.16
2	AQ-PM	91.40±0.70	3.6	51.37	6.92/11.16
	AQ-BPL	94.76±0.42	4.55	92.0	7.0/11.41
3	AQ-PM	93.35±0.69	4.41	58.95	7.94/13.27
	AQ-BPL	96.92±0.20	6.5	137.0	8.12/14.16
4	AQ-PM	94.75±0.54	5.27	66.11	8.78/14.88
	AQ-BPL	98.16±0.13	8.23	180.0	9.17/16.44
5	AQ-PM	95.98±0.34	6.29	73.11	9.97/17.25
	AQ-BPL	99.01±0.04	10.42	223.0	10.49/19.46

*$p < .01$

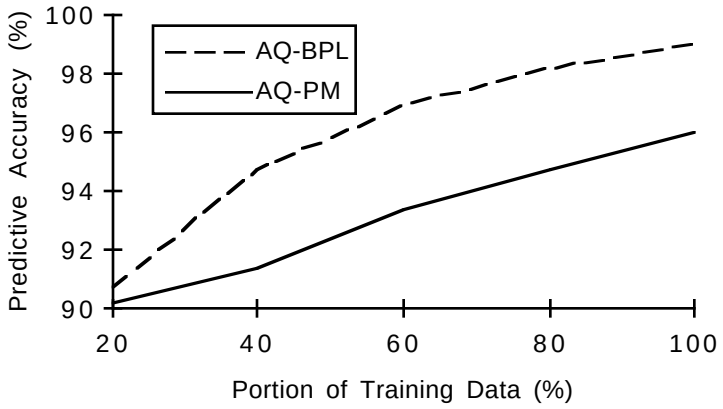


Figure 10: Predictive accuracy for the email learning agent.

The actual memory requirements for each learner can be seen in a comparison of the number of examples maintained (Figure 12). As with the computer intrusion detection problem, the number of training examples maintained by the partial memory approach was notably fewer than the number maintained by the baseline progressive learner. Specifically, AQ-BPL at partition 5 maintained 205% more training examples than AQ-PM (223 vs. 73.11).

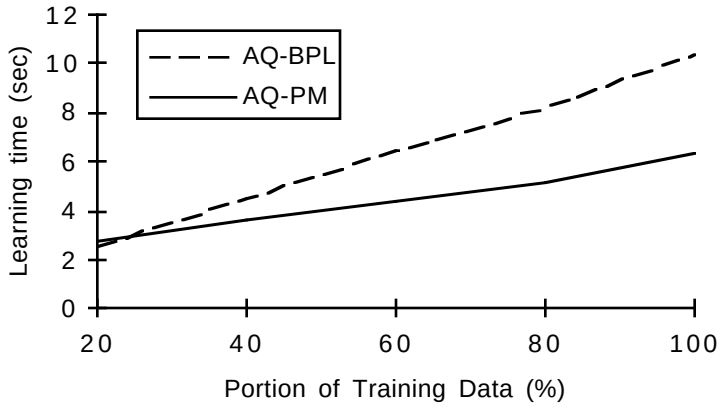


Figure 11: Learning times for the email learning agent.

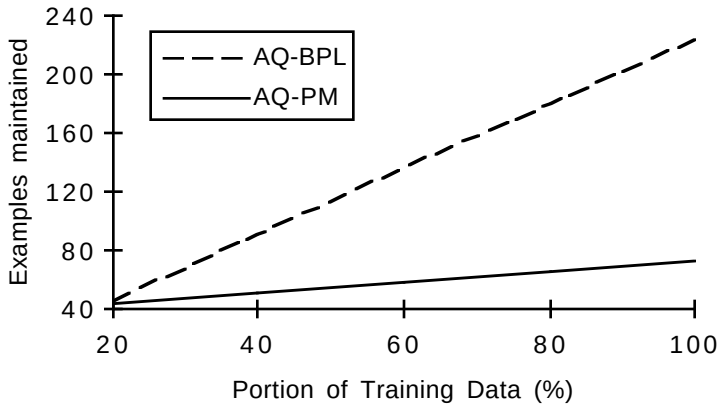


Figure 12: Memory requirements for the email learning agent.

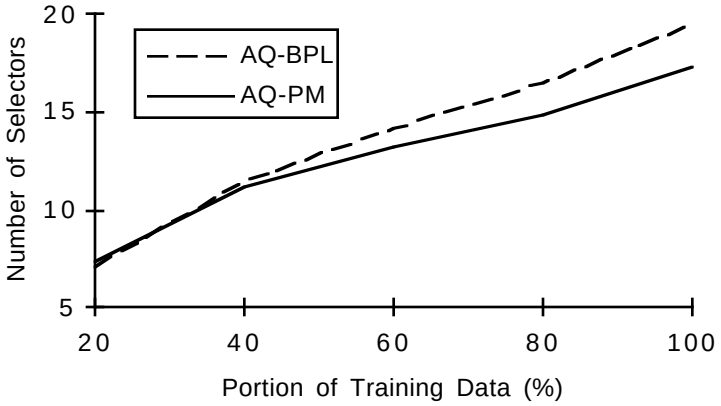


Figure 13: Concept complexity for the email learning agent.

The final comparison between these two learning methods for the email learning agent problem is using concept complexity. AQ-PM produced slightly less complex concepts than did AQ-BPL, as shown in Figure 13.

Figure 14 shows the rules induced for the uninteresting email class (UI). The first rule states that an email is uninteresting if it is sent by either maloof or jsmith, the email was a reply, and the email is short (i.e., the number of lines is less than 12). This rule makes intuitive sense. The agent's owner, who happens to be the author, finds his own emails uninteresting, especially if they are short replies. This rule handles cases in which the user carbon copied himself on a reply.

The presence of jsmith in the first uninteresting rule is the result of an example incorrectly classified by the user. The remaining rules are as intuitive, although there may have been additional incorrectly classified examples that crept into the training set. For example, one would hope that most people would find emails with the subject "Request" interesting and honor the request, if reasonable. But for some reason, this subject has been deemed uninteresting. This, however, illustrates one of the advantages of rule learning systems. Because the rules produced by AQ-PM are easily understood, it would be easy for anyone to read the rules and remove any unwanted rule conditions or training examples that produced those rule conditions.

The last rule in this concept description is very light, since it covers only 1 training example. The user may decide to remove this rule from the agent's knowledge-base. Learning systems that do not produce concept descriptions, such as *k*-nn (Aha et al. 1991), or produce nonsymbolic concept descriptions, such as neural networks (Rumelhart et al. 1986), make such post-learning analysis of the concepts difficult, if not impossible.

```
uninteresting  <::  [senderName = maloof@aic.gmu.edu ∨
                                  jsmith@dais.com] &
                    [reply = 1] & [numberOfLines = 0..12]
                    (t-weight: 6, u-weight: 6)
uninteresting  <::  [subject = “Request” ∨ “Colloquium Time” ∨
                               “US” ∨ “Away from my Mail” ∨
                               “Are you getting this” ∨
                               “returned mail for colloquium” ∨
                               “returned mail user unknown” ∨
                               “cs shutdown reminder”]
                    (t-weight: 4, u-weight: 3)
uninteresting  <::  [subject = “Tuesday Lunch” ∨
                               “Mac Printer Name Change”]
                    (t-weight: 1, u-weight: 1)
```

Figure 14: Examples of ELA rules for uninteresting emails.

6.4 Detection of Blasting Caps in X-Ray Images

Progressive learning is important for computer vision in a variety of contexts. From a systems development standpoint, an progressive learning system can acquire and gradually refine object descriptions through interaction with an expert. From an active vision perspective (Aloimonos et al. 1988), as an agent moves through an environment, it will continually perceive new views of known objects and will need to update its current set of concepts to accommodate these new views. From a video surveillance perspective, learning over time can be used to learn to track contours in an image sequence (Blake et al. 1995). Alternatively, the system may need to learn what motions are interesting versus those that are not (Duric et al. 1996).

6.4.1 Problem Description

Consider the problem of detecting blasting caps in x-ray images of airport luggage (Maloof and Michalski 1995d, 1996; Maloof et al. 1996). Although there will be a period of initial training before system deployment, it is doubtful that this period of training will be sufficient for the life of the system. Consequently, the system must possess the ability to continually learn once it is deployed.

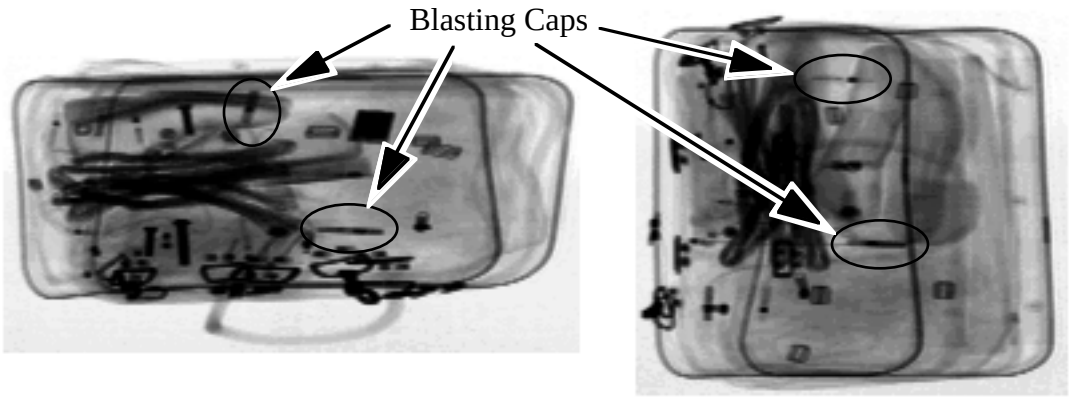


Figure 15: Examples of x-ray images of luggage containing blasting caps and clutter.

For this problem, a set of five x-ray images were selected from a collection of 30 images. Each image was of a suitcase containing blasting caps and varying amounts of clutter, such as shoes, calculators, pens, and the like. The selected images were of low to moderate complexity in terms of degree of clutter and positional variability of the blasting cap. Figure 15 shows two of the images used. Blasting cap and non-blasting cap objects were extracted from each of the images and characterized using 25 intensity, shape, and positional features, described in Table 4. These features were computed from the characteristic blob produced by the secondary high explosive and the rectangular region produced by the metal body of the blasting cap, as shown in Figure 16.

The SCALE implementation (Bloedorn et al. 1993) of the ChiMerge algorithm (Kerber 1992) was used to abstract the representation space into a smaller, discrete space. This resulted in attribute levels ranging from 4 to 14. The PROMISE score (Baim 1988) was computed to select the 15 most relevant attributes (i.e., those attributes achieving a score greater than 0.9). These attributes are designated in Table 4 by check marks (√). The training set consisted of 66 examples divided among two classes corresponding to blasting

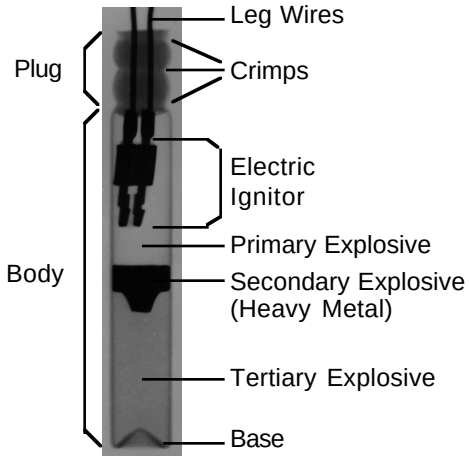


Figure 16: Detailed x-ray of a blasting cap.

caps and non-blasting caps. The size of the representation space was 2.6×10^{12} with the training examples representing 2.5×10^{-9}% of the space. Learning was conducted using 10 partitions of the data. For each partition, 100 learning and testing runs were conducted. Measures computed were predictive accuracy, learning time, concept complexity, and the number of examples maintained. The parameter settings used for this problem appear in Appendix C.

6.4.2 Experimental Results

The numerical results from the experiments for the blasting cap detection problem appear in Table 5. As before, AQ-PM and AQ-BPL, the baseline progressive learner, were compared using predictive accuracy, learning time, the number of examples maintained, and concept complexity.

The graphical summary of the predictive accuracy comparison, pictured in Figure 17, shows that AQ-PM performed well, achieving 93.13%, but not as well as AQ-BPL, which achieved 100%. The gap between the predictive accuracies for the learners in

Table 4: Attributes used to represent blasting caps.

	Attribute	Description
	blobMinimum	Minimum Blob Intensity
√	blobMaximum	Maximum Blob Intensity
√	blobAverage	Average Blob Intensity
√	blobLength	Blob Length
	blobWidth	Blob Width
	blobArea	Blob Area
	blobPerimeter	Blob Perimeter
√	blobCompactness1	$(4\pi * \text{blobArea}) / \text{blobPerimeter}^2$
√	blobCompactness2	blobAverage / (max(blobLength, blobWidth) + 1)
√	blobCompactness3	blobCompactness1 * blobAverage
	blobIntensitySD	Standard Deviation of Blob Intensity
	rectangleMinimum	Minimum Rectangle Intensity
	rectangleMaximum	Maximum Rectangle Intensity
	rectangleAverage	Average Rectangle Intensity
√	rectangleLength	Rectangle Length
√	rectangleWidth	Rectangle Width
√	rectangleArea	Rectangle Area
	rectanglePerimeter	Rectangle Perimeter
√	rectangleCompactness1	$(4\pi * \text{rectangleArea}) / \text{rectanglePerimeter}^2$
√	rectangleCompactness2	rectangleAverage / (max(rectangleLength, rectangleWidth) + 1)
√	rectangleCompactness3	rectangleCompactness1 * rectangleAverage
√	rectangleIntensitySD	Standard Deviation of Rectangle Intensity
√	centroidDistances	Distances between blob and rectangle centroids
√	verticalCentroidDistance	Vertical component of centroidDistances
	horizontalCentroidDistance	Horizontal component of centroidDistances

√ Denotes relevant attribute

partition 10 is greater for this problem (i.e., a 7% gap) than for the previous two problems. The possible causes for this drop in performance, comparing to the two previous experiments, is discussed in Section 7.4.

As in previous experiments, AQ-PM yielded faster learning times than AQ-BPL, as shown in Figure 18. For partition 10, AQ-BPL ran about 113% slower than AQ-PM (2.45s vs. 1.15s). Again, this increase in learning speed is due to the partial memory learning approach.

Table 5: Experimental results for the blasting cap detection problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)*	Learning Times (sec)	Examples Maintained	Concept Complexity (rules/selectors)
1	AQ-PM	67.36±2.40	0.3	7.0	2.01/2.09
	AQ-BPL	67.36±2.40	0.29	7.0	2.01/2.09
2	AQ-PM	80.38±1.58	0.39	9.6	2.05/3.0
	AQ-BPL	83.73±1.39	0.48	14.0	2.15/3.69
3	AQ-PM	84.65±1.28	0.48	11.21	2.25/3.78
	AQ-BPL	89.14±0.96	0.69	21.0	2.51/5.09
4	AQ-PM	86.03±1.38	0.55	12.68	2.41/4.47
	AQ-BPL	92.56±0.64	0.99	28.0	2.99/6.36
5	AQ-PM	87.45±1.50	0.67	14.06	2.65/5.18
	AQ-BPL	95.19±0.50	1.27	35.0	3.44/7.63
6	AQ-PM	88.77±1.26	0.77	15.25	2.82/5.61
	AQ-BPL	95.95±0.46	1.51	42.0	3.73/8.6
7	AQ-PM	90.22±1.20	0.86	16.43	3.01/6.33
	AQ-BPL	97.36±0.32	1.8	49.0	4.08/9.61
8	AQ-PM	90.80±1.11	1.0	17.46	3.26/6.84
	AQ-BPL	98.10±0.30	2.04	55.0	4.38/10.67
9	AQ-PM	91.33±0.98	1.09	18.29	3.41/7.25
	AQ-BPL	99.24±0.21	2.28	61.0	4.72/11.78
10	AQ-PM	93.13±0.96	1.15	19.0	3.55/7.64
	AQ-BPL	100.00±0.00	2.45	66.0	5.0/12.33

*$p < .01$

With regards to memory requirements, Figure 19 shows the improved memory requirements of AQ-PM over AQ-BPL for the blasting cap recognition problem. For partition 10, AQ-PM maintained 19 representative examples, while AQ-BPL maintained 66 training examples, on average. At this point in learning, AQ-BPL maintained roughly 247% more examples than AQ-PM.

The final comparison between AQ-PM and AQ-15c for the blasting cap detection problem is concept complexity. In previous experiments, the difference in concept complexity between AQ-PM and AQ-BPL has been negligible. For this experiment, however, concepts produced by AQ-PM were noticeably simpler than the concepts produced by AQ-BPL, as shown in Figure 20. The possible causes for this are discussed in Section 7.4.

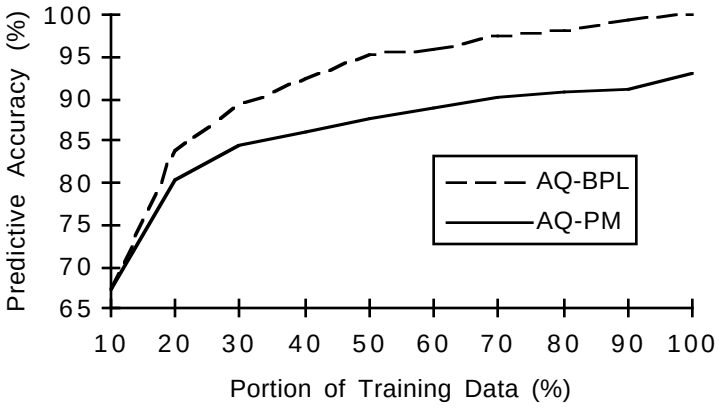


Figure 17: Predictive accuracy for the blasting cap detection problem.

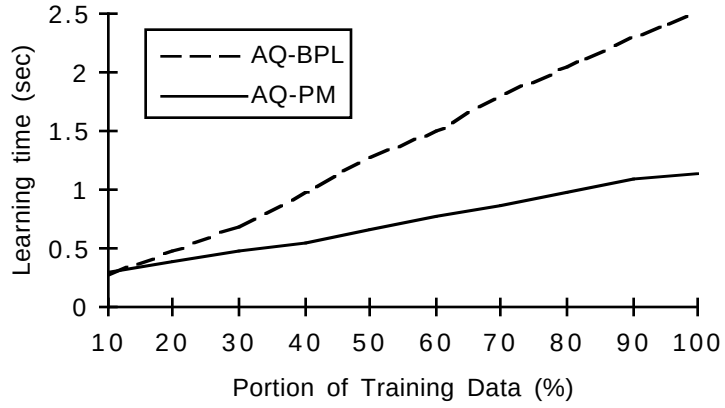


Figure 18: Learning times for the blasting cap detection problem.

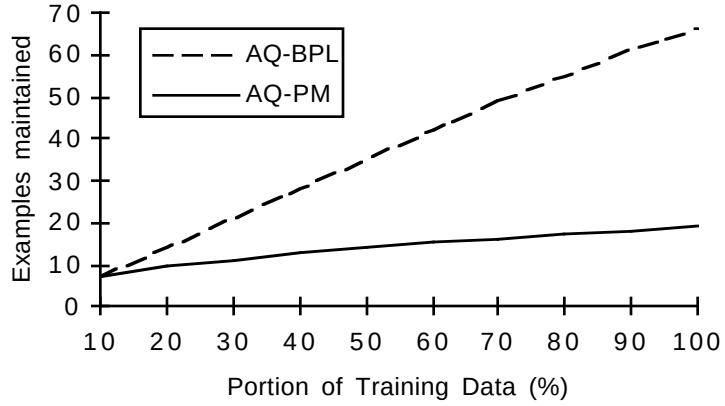


Figure 19: Memory requirements for the blasting cap detection problem.

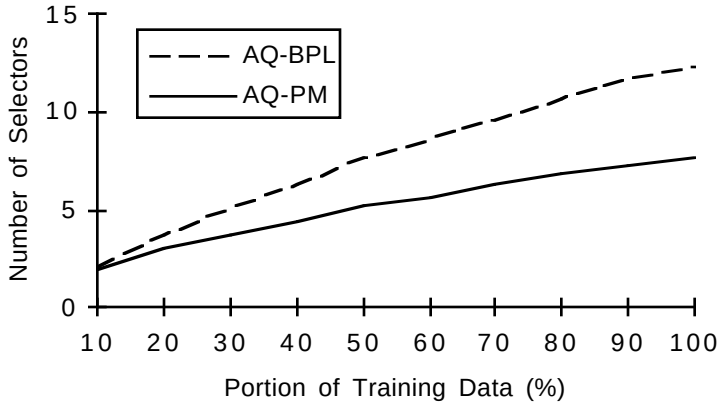


Figure 20: Concept complexity for the blasting cap detection problem.

Figure 21 shows concept descriptions for a blasting cap and a non-blasting cap. The more intuitive and interesting concept description is for the blasting cap. Recall from Figure 16 the components of an x-rayed blasting cap. The first rule for a blasting cap states that the blasting cap length (i.e., `rectangleLength`) is between 18 and 51 pixels long, the width (i.e., `rectangleWidth`) is between 0 and 8 pixels wide, and the center of the blob is located roughly near the center of the rectangular region (i.e., `centroidDistances`). Alternatively, the standard deviation of the intensities within the rectangular region (i.e., `rectangleIntensitySD`) is between 28.03 and 31.87. The first rule for the blasting cap concept is intuitive and descriptive of blasting caps. It expresses the salient characteristics of an x-rayed blasting cap: its length, its width, and the presence of the characteristic blob near the center of the body. This illustrates the importance and benefit of learning systems that produce comprehensible and understandable concept descriptions.

```
   cap  <::  [rectangleLength = 18.0..51.0] &
             [rectangleWidth = 0.0..7.99] &
             [centroidDistances = 0.0..8.49]
             (t-weight: 8, u-weight: 5)
   cap  <::  [rectangleIntensitySD = 28.03..31.87]
             (t-weight: 5, u-weight: 2)

noncap  <::  [rectangleWidth = 5.0..91.0] &
             [rectangleCompactness1 = 3.89..26.95]
             (t-weight: 15, u-weight: 11)
noncap  <::  [blobCompactness2 = 31.68..40.31]
             (t-weight: 6, u-weight: 2)
noncap  <::  [rectangleArea = 8.0..135.99] &
             [rectangleCompactness2 = 22.61..39.82]
             (t-weight: 2, u-weight: 2)
```

Figure 21: Examples of AQ-PM rules for blasting caps and non-blasting caps.

6.5 Changing Concepts: The STAGGER Concepts

The next set of experiments involves learning a changing concept. Although elements of change are certainly present in the other problems, because these problems were taken from the real world, it is difficult to quantify and understand the types of change present. Therefore, this section presents experiments with the STAGGER concepts (Schlimmer and Granger 1986; Widmer and Kubat 1996), a synthetic problem domain.

6.5.1 Problem Description

The representation space for the STAGGER concepts (Schlimmer and Granger 1986) consists of three attributes: shape, size, and color. Each attribute has three levels. Shape can be either square, circular, or triangular. Size can be either small, medium, or large. Color can be either red, green, or blue. The training set is randomly generated over a time period of 120 steps. One training example is presented each time step. For the first 40 time steps, the concept to be learned is [size = small] & [color = red], diagrammed in Figure 22a. For the next 40 time steps, the concept is [color = green] $\vee$ [shape = circular]

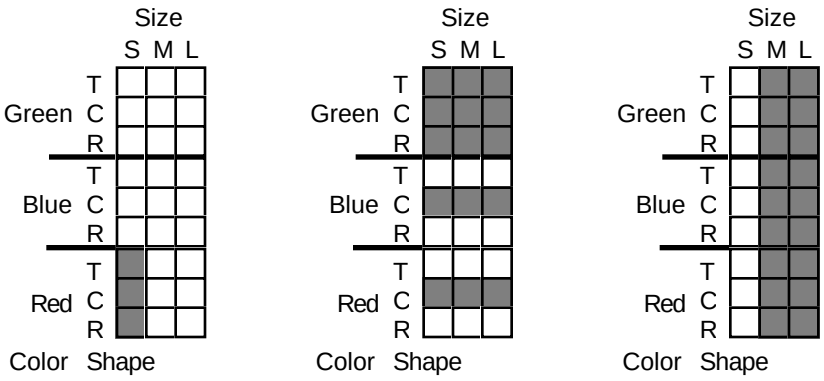
(Figure 22b). For the remaining time steps, the concept is [size = medium $\vee$ large] (Figure 22c).

At each time step, the current set of concepts is validated against a 100 event test set, also randomly generated using the appropriate concepts for the current time step. One slight modification was made to Widmer and Kubat's (1996) experimental protocol. For the first time step, both AQ-PM and AQ-BPL were permitted to learn from two examples, one from each class. The reason for this was that the AQ15c system (Wnek et al. 1995) is presently configured to learn from one or more training examples in each class. Although it is possible for AQ to learn from one or more examples from one class using the refunion operator, we chose to provide AQ with an extra training example. The 120 time step experiment was repeated 10 times. Only predictive accuracy results are presented. This experimental design is the same used by Widmer and Kubat (1996). The parameters used for this problem appear in Appendix D.

6.5.2 Experimental Results

Figure 23 shows how AQ-PM learns a changing concept. The first 40 time steps looks much like previous learning curves with AQ-BPL performing slightly better than AQ-PM. At the 40th time step, however, the concept changes and the predictive accuracy of both systems plummets. Another concept change occurs at time step 80 which results in another drop in predictive accuracy. AQ-PM is able to recover more readily than AQ-BPL to concept change because it is not burdened with past examples.

The AQ-PM results presented for the STAGGER concepts are comparable to the performance of STAGGER (Schlimmer 1987a) and the FLORA systems (Widmer and Kubat 1996). It is difficult to directly compare AQ-PM's performance to STAGGER, since Schlimmer and Granger (1986) used a slightly different experimental design.



(a) Target concept for time steps 1–39.

(b) Target concept for time steps 40–79.

(c) Target concept for time steps 80–120.

Figure 22: Visualization of the STAGGER concepts.

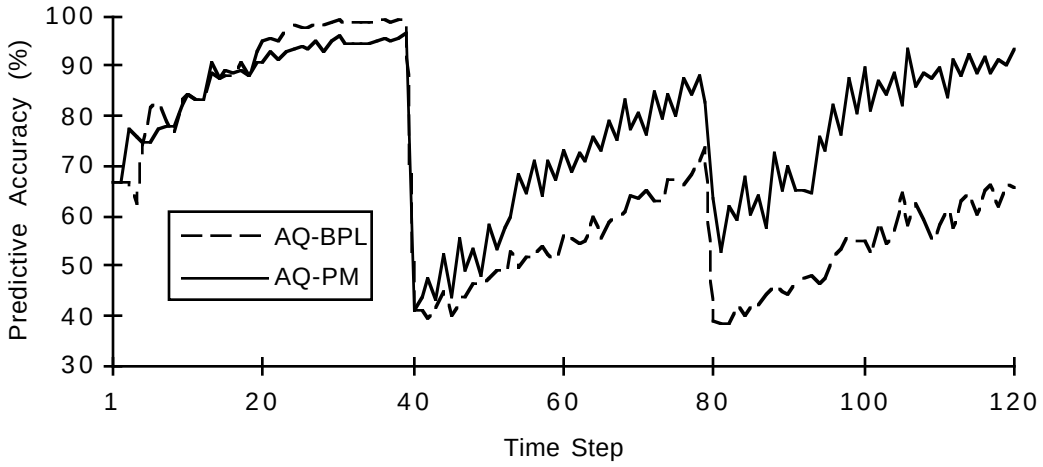


Figure 23: Tracking concept drift using the STAGGER concepts.

Widmer and Kubat's (1996) design was preferred and used because it was more recent and provided enough details for replication.

Unfortunately, this experimental design, i.e., presentation of the STAGGER concepts repeated 10 times, did not produce statistically significant results for many of the time steps. Consequently, a second experiment was conducted using the STAGGER concepts in which 120 presentations were performed. The results for this experiment are plotted in Figure 24. For all time steps, with the exception of time steps 1, 7, 8, 9, and 10, results are statistically significant at $p < .05$.

The reasons for the difficulty in achieving statistical significance for these runs is due to several factors. Notice in Figure 24 that at time steps 7–10 the predictive accuracy curves for AQ-BPL and AQ-PM intermingle and cross. The performances of these two similar algorithms, AQ-PM and AQ-BPL, are essentially the same at these points. The factors that influence statistical significance are the algorithms being evaluated, the amount of data used in each learning run, and the number of learning runs at each time step. Statistical significance is difficult to achieve for the STAGGER experiments because we are comparing virtually the same algorithm (i.e., AQ15c with and without modifications), using very little data (i.e., at each time step only one new training example is presented). Therefore our only variable is the number of learning runs. In spite of numerous repeated learning runs, statistically, we cannot that say that for time steps 7–10 that the performance differences between AQ-BPL and AQ-PM are not due to other causes of variation. When we look at the experiment as a whole, however, and recognize that statistical significance was achieved during all other runs, then it is easy to infer that the inability to achieve statistical significance for time steps 7–10 is not due to outside factors, but it is due to a period of similar performance by the two algorithms. Note also that a result that is not statistically significant does not imply that it is insignificant (Keppel et al. 1992).

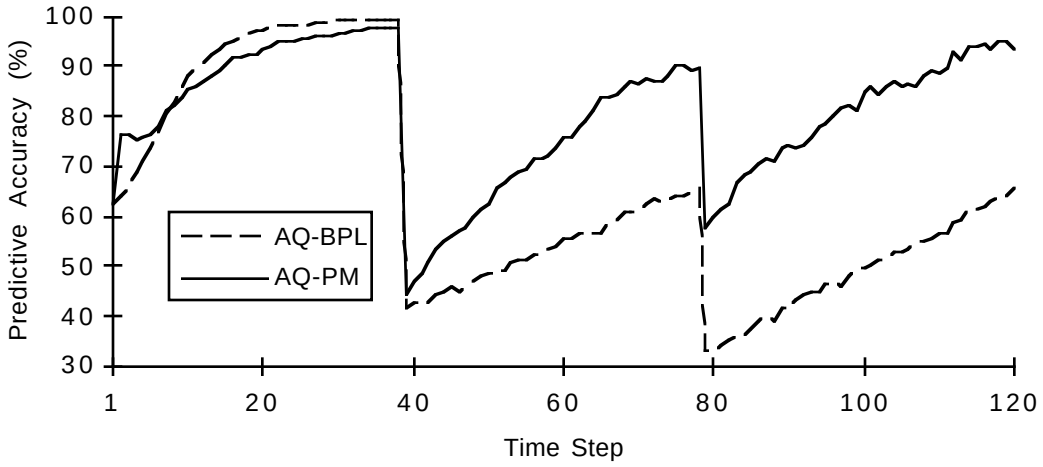


Figure 24: Results from the second experiment using the STAGGER concepts.

Comparing more precisely to Widmer and Kubat's (1996) results, shown in Figure 25 (Widmer 1996b), AQ-PM's performance is comparable to the performances of the FLORA systems with a few exceptions. In the first time period (i.e., 1–40), the FLORA systems were able to converge to the target concept more quickly than AQ-PM. At time step 39, the predictive accuracies for FLORA2, 3, and 4, and AQ-PM were 97, 97, 100, and 97%, respectively. After the concept shift at time step 40, the performance of the FLORA learning systems drops to 47% with AQ-PM's performance slightly lower at 44%.

For time steps 40–80, FLORA4's predictive accuracy climbed more quickly than the other systems. At time step 79, just before another concept shift, the predictive accuracies for FLORA2, 3, and 4, and AQ-PM were 98, 99, 98, and 90%, respectively. After the concept shift at time step 80, the predictive accuracies of the four algorithms drops again to 55, 56, 55, and 58% for FLORA2, 3, and 4, and AQ-PM.

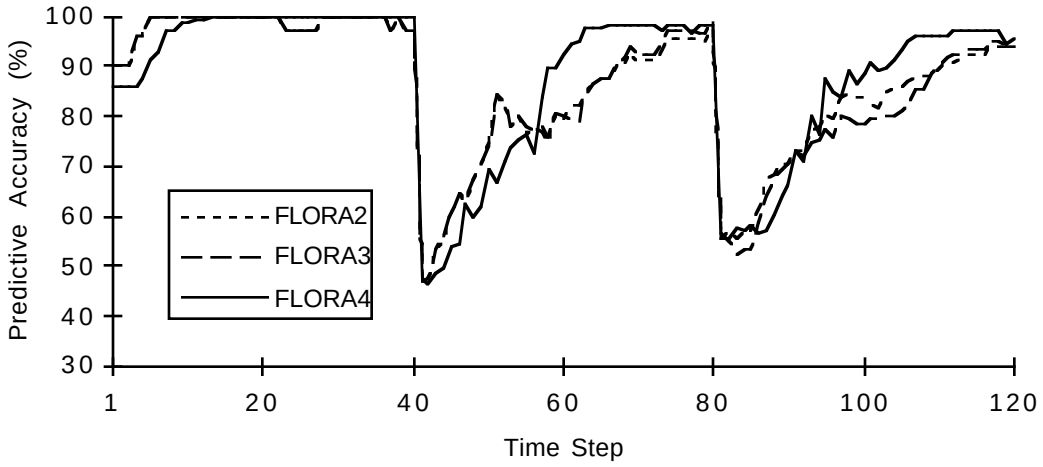


Figure 25: FLORA results using the STAGGER concepts (Widmer 1996b).

For time steps 80–120, as before, FLORA4 was able to converge more quickly to the target concept than the other learning systems. At time step 120, the predictive accuracies for the four algorithms are nearly the same: FLORA2 achieved 94%, FLORA3 94%, FLORA4 96%, and AQ-PM 95%.

Comparing predictive accuracy performance, AQ-PM did not, perhaps, do quite as well as the FLORA systems. It converged more slowly during time steps 1–40, and did not achieve as high predictive accuracies during time steps 40–80. On the other hand, for time steps 80–120, the performance of all systems were comparable. These results, however, are best examined in the context of memory requirements.

Figure 26 illustrates the number of examples maintained by AQ-PM and FLORA2 for the STAGGER experiment. Overall, FLORA2 maintained, on average, over a 120 time step experiment, 116% more examples than AQ-PM. Note that results for memory requirements were presented only for FLORA2 (Widmer and Kubat 1996). Nevertheless, FLORA3 and 4 use the same memory management mechanisms as FLORA2, so it is anticipated that the memory requirements for FLORA3 and 4 were similar. In time steps 1–

40, as each system's predictive accuracy reached a high level of performance, the number of examples maintained by the system stabilized. At time step 39, FLORA2 maintained 15 examples, while AQ-PM maintained roughly 6 examples. Once the shift in concept had occurred at time step 40, both systems reacted by increasing the number of examples maintained. At time step 50, FLORA2 rose to a level of 24 examples and AQ-PM rose to a level of 9 examples. Notice that after both systems had increased the number of examples maintained, both forgot the older examples. During the time period 50–80, FLORA2's memory requirements fluctuate, while AQ-PM stabilizes at roughly 7 examples. Another concept shift occurs at time step 80 and again, both systems react. This time, AQ-PM increased the number of examples maintained, as in time step 40, but FLORA2 forgets several examples, before increasing its memory size to 22 examples and then dropping to 21 examples. After time step 90, AQ-PM's memory size decreases until the experiment stops, which is a result of the time-based forgetting parameter (see Appendix D).

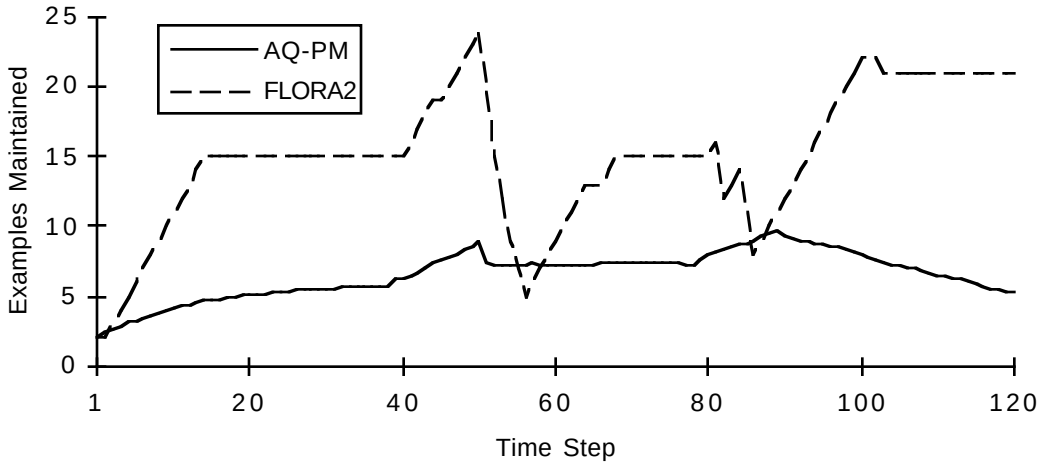


Figure 26: Memory requirements for AQ-PM and FLORA2 for the STAGGER experiment.

Figure 27 shows the set of concept descriptions produced by AQ-PM at time step 39, which is before the first concept change occurs. The target concept for this time period is [size = small] & [color = red], which AQ-PM learned.

For the next time period, from time step 40 to 79, the target concept is [color = green] $\vee$ [shape = circular]. Again, AQ-PM was able to learn the target concept. As shown in Figure 28, which is the concept description at time step 79, again just before the second concept shift. Notice that there are remnants of the previous concept description still present in the negative concept description, specifically, the selector [size = small $\vee$ medium].

For the final time period, from time step 80 to 120, the target concept is [size = medium $\vee$ large]. As shown in Figure 29, which is the concept description at the final time step 120, AQ-PM again learned the correct concept for the positive class.

6.6 Comparison of AQ-PM to AQ11 and GEM

To place the AQ-PM experimental results within the context of past incremental learning work, experimental comparisons were made to AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988). The features of these learning systems are integrated into the AQ15 inductive learning system (Hong et al. 1986). Recall from Chapter 3 on related research that AQ11 is an incremental learning variant of the AQ algorithm (Michalski 1969) using a no memory model, while GEM is an AQ-variant using a full memory model. For this comparison, the blasting cap detection problem and a reduced version of the intrusion detection problem were repeated using 50 learning and testing runs for each 10% partition of the original data set. Testing was conducted on the entire data set. Again, the measures computed were predictive accuracy, learning time, concept complexity, and the number of examples maintained.

```
pos  <::  [size = small] & [color = red]
          (t-weight: 1, u-weight: 1)

neg  <::  [size = medium ∨ large]
          (t-weight: 4, u-weight: 3)
neg  <::  [color = blue ∨ green]
          (t-weight: 2, u-weight: 1)
```

Figure 27: Examples of AQ-PM rules for the STAGGER concepts at time step 39.

```
pos  <::  [color = green]
          (t-weight: 3, u-weight: 3)
pos  <::  [shape = circular]
          (t-weight: 2, u-weight: 2)

neg  <::  [size = small] & [color = red]
          (t-weight: 2, u-weight: 2)
neg  <::  [color = blue] &
          [shape = triangular]
          (t-weight: 1, u-weight: 1)
neg  <::  [size = small ∨ medium] &
          [color = blue]
          (t-weight: 1, u-weight: 1)
```

Figure 28: Examples of AQ-PM rules for the STAGGER concepts at time step 79.

```
pos  <::  [size = medium ∨ large]
          (t-weight: 3, u-weight: 3)

neg  <::  [size = small]
          (t-weight: 2, u-weight: 2)
neg  <::  [size = large] &
          [color = blue ∨ green]
          (t-weight: 1, u-weight: 1)
```

Figure 29: Examples of AQ-PM rules for the STAGGER concepts at time step 120.

The intrusion detection data set was reduced by removing three attributes (average real time, average CPU time, and maximum hog factor) that had more than 58 attribute values. AQ11 and GEM, which are integrated in the AQ15 system (Hong et al. 1986), have an upper limit of 58 attribute values due to a limitation in the Pascal compiler. This limitation is not present in AQ15c (Wnek et al. 1995), the C implementation of AQ15. See Section 6.2 for complete details about the computer intrusion detection problem. See Section 6.4 for details about the blasting cap detection problem.

6.6.1 Experimental Results for the Blasting Cap Data Set

The numerical results from the experiments comparing AQ-PM, AQ11, and GEM for the blasting cap detection problem appear in Table 6. AQ-PM, AQ11, and GEM were compared using predictive accuracy, learning time, the number of examples maintained, and concept complexity.

The graphical summary of the predictive accuracy comparison, pictured in Figure 30, shows that AQ-PM performed well, achieving 93%, but not as well as AQ11 and GEM. AQ11 achieved 100%, while GEM achieved 99%. For this problem, the predictive accuracy performance of AQ11 and GEM resembles the predictive accuracy performance of the AQ15c baseline progressive learner (see Figure 17). The difference in predictive accuracy between AQ-PM and the other the learners in partition 10 is 6–7%, for this problem. Note that, in terms of predictive accuracy, this was the worst problem for AQ-PM.

Figure 31 shows the learning times for the three learners. AQ-PM shows a steady increase in learning time as more training data is processed. The learning times of both AQ11 and GEM are similar to AQ-PM's learning time after 40% of the data has been presented. The learning times of AQ11 and GEM fall as concept descriptions are refined toward the target concept. Note also that much of the variation in the learning time is likely

Table 6: Experimental results for the blasting cap detection problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)	Learning Times (sec)	Examples Maintained	Concept Complexity (rules/selectors)
1	AQ-PM	67.36±2.40	0.3	7.0	2.01/2.09
	GEM	68.68±1.63	3.42	7.0	2.0/2.0
	AQ11	65.75±1.64	3.55	0	2.0/2.0
2	AQ-PM	80.38±1.58	0.39	9.6	2.05/3.0
	GEM	83.01±1.23	1.44	14.0	2.69/3.87
	AQ11	82.35±1.03	1.6	0	2.7/3.72
3	AQ-PM	84.65±1.28	0.48	11.21	2.25/3.78
	GEM	87.57±0.91	0.7	21.0	3.54/5.67
	AQ11	88.05±0.79	0.95	0	3.8/6.6
4	AQ-PM	86.03±1.38	0.55	12.68	2.41/4.47
	GEM	90.17±0.7	0.56	28.0	4.14/7.1
	AQ11	90.41±0.69	0.6	0	4.5/8.3
5	AQ-PM	87.45±1.50	0.67	14.06	2.65/5.18
	GEM	92.38±0.67	0.69	35.0	4.6/8.51
	AQ11	92.73±0.59	0.8	0	6.11/13.17
6	AQ-PM	88.77±1.26	0.77	15.25	2.82/5.61
	GEM	94.24±0.59	0.57	42.0	4.99/9.88
	AQ11	94.35±0.52	0.76	0	5.2/10.4
7	AQ-PM	90.22±1.20	0.86	16.43	3.01/6.33
	GEM	95.84±0.54	0.87	49.0	5.87/11.6
	AQ11	95.82±0.42	0.96	0	6.7/15.5
8	AQ-PM	90.80±1.11	1.0	17.46	3.26/6.84
	GEM	97.05±0.41	0.77	55.0	5.87/12.63
	AQ11	97.35±0.3	1.35	0	8.12/19.66
9	AQ-PM	91.33±0.98	1.09	18.29	3.41/7.25
	GEM	98.3±0.28	0.96	61.0	6.4/14.5
	AQ11	98.64±0.22	1.32	0	7.78/18.41
10	AQ-PM	93.13±0.96	1.15	19.0	3.55/7.64
	GEM	99.14±0.24	0.82	66.0	6.63/15.56
	AQ11	100.0±0.0	1.37	0	8.4/21.99

due to implementation. Both AQ11 and GEM are written in Pascal, while AQ-PM, which is based on AQ15c (Wnek et al. 1995), is written in C. Although AQ11 and GEM are predecessors of AQ15c, AQ15c has undergone several iterations of performance optimizations which could account for differences in performance.

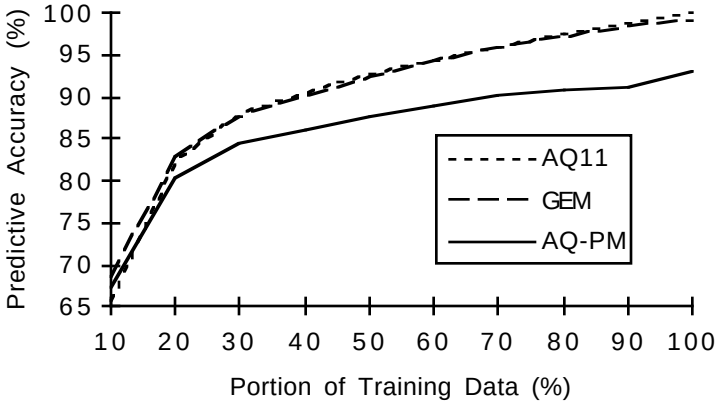


Figure 30: Predictive accuracy for the blasting cap detection problem.

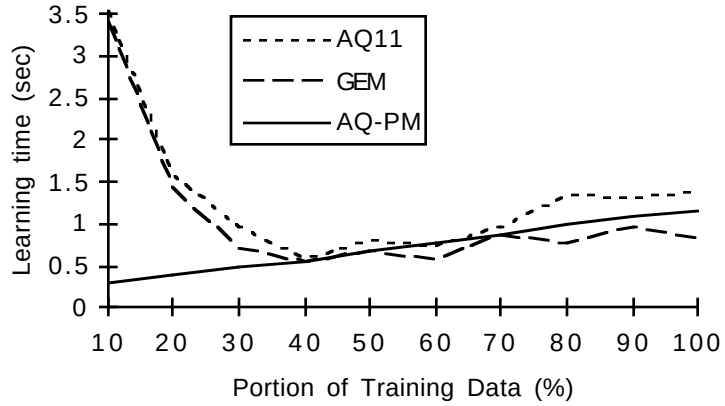


Figure 31: Learning times for the blasting cap detection problem.

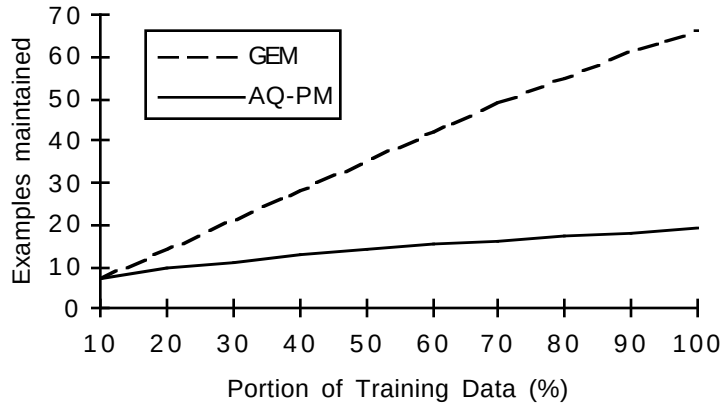


Figure 32: Memory requirements for the blasting cap detection problem.

Figure 32 shows the memory requirements for GEM and AQ-PM. Since GEM is a full memory incremental learner, it maintains the same number of examples as the baseline progressive learner, which is substantially more than the AQ-PM maintains. On the other hand, since AQ11 is a no memory incremental learner, it maintains no examples. This is why no line for AQ11 appears in Figure 32. When learning ended at partition 10, GEM maintained roughly 247% more examples than AQ-PM (66 vs. 19).

The final comparison between AQ-PM, AQ11, and GEM is concept complexity, shown in Figure 33. AQ-PM's concept descriptions were much less complex than both AQ11 and GEM. GEM's concepts, on average, consisted of 103% more selectors than did AQ-PM's (15.56 vs. 7.64) at partition 10. AQ11's concepts, on average, consisted of 187% more selectors than did AQ-PM's (21.99 vs. 7.64) at partition 10. Additionally, looking at how AQ11's concept complexity varies over time, not only does the number of selectors increase, it also fluctuates. This is evidence of concept instability (O'Rourke 1982), although the effect is dampened by averaging the selector counts over 100 learning runs. The issue of concept instability is discussed further in Section 7.4. Examples of rules for blasting caps produced by GEM and AQ11 are shown in Figures 34 and 35, respectively. Examples of rules produced by AQ-PM are shown in Figure 21.

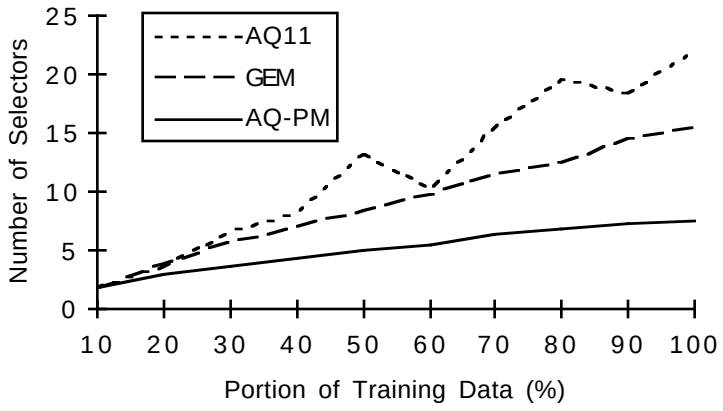


Figure 33: Concept complexity for the blasting cap detection problem.

```
cap  <::  [blobMaximum <> 42.0..60.99] &
          [blobLength <> 7.0..9.99] &
          [rectangleWidth <> 8.0..91.00] &
          [blobCompactness2 <> 36.9..40.31 ∨
          43.73..46.02 ∨ 47.44..58.59] &
          [rectangleIntensitySD <> 15.18..21.84 ∨
          24.33..28.02 ∨ 37.53..38.49 ∨ 44.74..54.95]
          (t-weight: 22, u-weight: 9)
cap  <::  [blobCompactness1 = 3.9..6.14] &
          [blobCompactness2 <> 36.9..40.31 ∨
          43.73..46.02]
          (t-weight: 10, u-weight: 4)
cap  <::  [blobAverage = 8.0..24.99] &
          [blobLength <> 3.0..4.99]
          (t-weight: 7, u-weight: 2)
cap  <::  [blobMaximum <> 42.0..60.99] &
          [blobCompactness2 <> 36.9..40.31 ∨
          43.73..46.02] &
          [rectangleArea = 157.00..172.99]
          (t-weight: 6, u-weight: 1)
```

Figure 34: Examples of GEM rules for blasting caps.

```
cap  <::  [blobMaximum = 0.0..41.99 ∨ 61.0..131.99] &
          [blobCompactness2 = 4.71..18.45 ∨
          24.86..31.67 ∨ 36.15..36.89 ∨ 40.32..43.72 ∨
          46.03..47.43 ∨ 58.6..87.77]
          (t-weight: 25, u-weight: 4)
cap  <::  [blobCompactness2 = 4.71..18.45 ∨
          24.86..31.67 ∨ 36.15..36.89 ∨ 40.32..43.72 ∨
          46.03..47.43 ∨ 58.6..87.77] &
          [verticalCentroidDistance <> 4.4..5.69 ∨
          8.5..100.0]
          (t-weight: 23, u-weight: 0)
cap  <::  [blobCompactness2 = 4.71..31.67 ∨
          36.15..36.89 ∨ 40.32..43.72 ∨ 46.03..47.43 ∨
          58.6..87.77] &
          [verticalCentroidDistance = 5.7..8.49]
          (t-weight: 12, u-weight: 2)
```

Figure 35: Examples of AQ11 rules for blasting caps.

6.6.2 Experimental Results for Computer Intrusion Detection Problem

The numerical results from the experiments comparing AQ-PM, AQ11, and GEM for a smaller version of the computer intrusion detection problem appear in Table 7. AQ-PM, AQ11, and GEM were compared using predictive accuracy, learning time, memory requirements, and concept complexity.

The graphical summary of the predictive accuracy comparison is pictured in Figure 36. In terms of predictive accuracy, GEM and AQ11 slightly outperformed AQ-PM. For the last step of learning, both GEM and AQ11 achieved 100% on testing data, while AQ-PM achieved 97%.

Figure 37 shows the learning times for the three learners. As with the comparison using the blasting caps data set, AQ-PM shows a steady increase in learning time as more training data is processed. The learning times of both AQ11 and GEM are similar to AQ-PM's learning time after about half of the data had been processed.

Figure 38 shows the memory requirements for GEM and AQ-PM. GEM, the full memory learner, shows a steady increase in memory requirements as more and more of the data is processed. AQ11, on the other hand, has no memory requirements, since it uses a no memory model. The memory requirements for AQ-PM are consistent with previous experiments. When learning ended at partition 10, GEM maintained roughly 316% more examples than AQ-PM (238 vs. 57.1).

The final comparison between AQ-PM, AQ11, and GEM is concept complexity, shown in Figure 39. Unlike the previous comparison using the blasting caps data set, the concept complexities for the three learners are similar. AQ11 still shows signs of concept instability (O'Rourke 1982) at the 5th partition. This is discussed further in Section 7.4. Examples of rules for coyote's computing behavior produced by GEM and AQ11 are shown in Figures 40 and 41. Examples of rules for coyote's computing behavior produced by AQ-PM are shown in Figure 9.

Table 7: Experimental results for the computer intrusion detection problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)	Learning Times (sec)	Examples Maintained	Concept Complexity (rules/selectors)
1	AQ-PM	80.14±1.27	4.7	26.88	9.48/11.16
	GEM	76.82±1.28	47.84	24.0	9.0/9.0
	AQ11	76.22±1.23	48.38	0	9.0/9.0
2	AQ-PM	86.6±0.93	5.9	33.24	10.44/14.62
	GEM	85.5±0.80	13.39	48.0	9.26/9.98
	AQ11	85.46±0.75	13.92	0	9.22/10.0
3	AQ-PM	90.0±0.83	7.0	38.4	11.54/17.44
	GEM	90.18±0.66	9.37	72.0	9.68/11.56
	AQ11	90.3±0.67	11.12	0	9.64/11.62
4	AQ-PM	91.64±0.89	7.73	42.54	12.42/19.42
	GEM	93.04±0.57	5.72	96.0	10.06/13.14
	AQ11	92.64±0.62	5.92	0	10.44/14.52
5	AQ-PM	92.53±0.93	8.52	46.04	13.34/21.72
	GEM	95.7±0.49	5.55	120.0	10.62/15.34
	AQ11	95.48±0.53	8.31	0	12.28/22.8
6	AQ-PM	93.81±0.68	9.07	49	14.02/23.48
	GEM	96.88±0.4	4.69	144.0	11.04/16.74
	AQ11	96.62±0.51	6.02	0	11.58/20.2
7	AQ-PM	94.75±0.76	9.55	51.44	14.4/24.88
	GEM	98.34±0.28	5.46	168.0	11.78/19.52
	AQ11	97.94±0.32	7.92	0	12.2/22.28
8	AQ-PM	96.19±0.71	9.85	53.44	14.88/25.52
	GEM	99.02±0.22	5.54	192.0	12.06/21.24
	AQ11	99.0±0.22	8.1	0	12.72/24.86
9	AQ-PM	96.45±0.62	10.41	55.52	15.52/27.22
	GEM	99.7±0.14	6.8	216.0	12.62/23.26
	AQ11	99.74±0.12	9.83	0	13.52/27.92
10	AQ-PM	97.14±0.59	10.91	57.1	15.96/28.22
	GEM	100.0±0.0	7.55	238.0	13.2/25.1
	AQ11	100.0±0.0	10.72	0	14.22/30.8

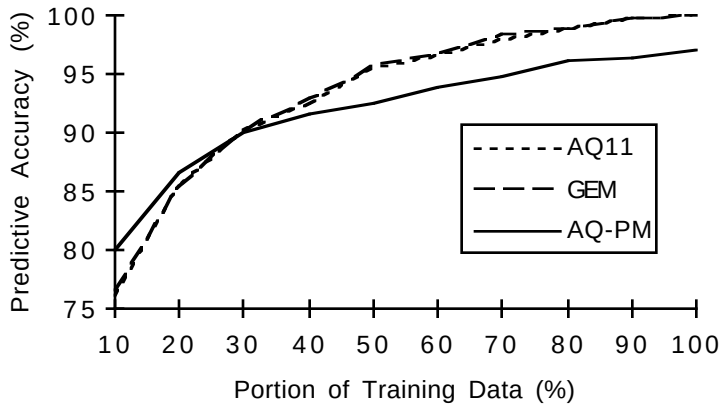


Figure 36: Predictive accuracy for the intrusion detection problem.

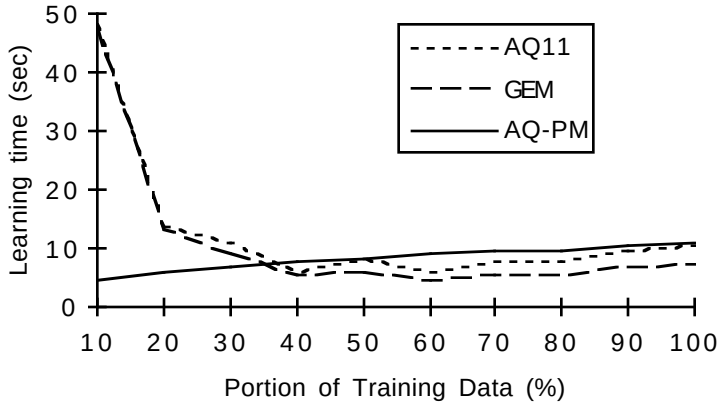


Figure 37: Learning times for the computer detection problem.

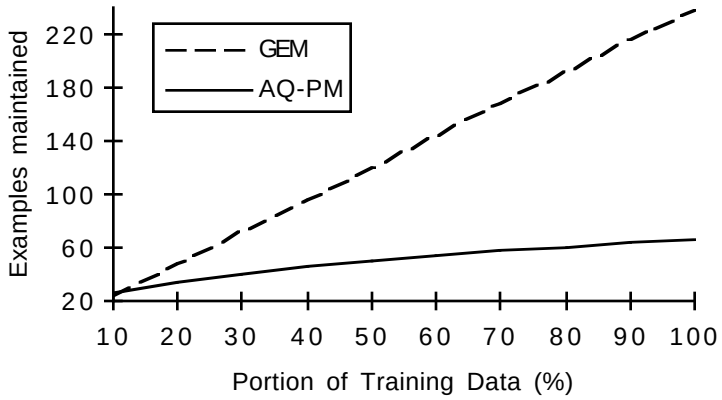


Figure 38: Memory requirements for the computer detection problem.

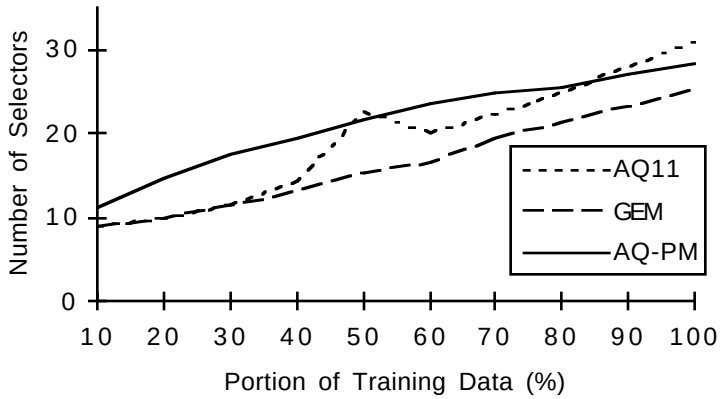


Figure 39: Concept complexity for the intrusion detection problem.

```
coyote  <::  [maximumSystemTime <> 0.05 ∨ 0.15..0..0.16 ∨
             0.2..0.23 ∨ 0.61..2.41] &
             [averageHogFactor = 0.0..0.84 ∨
             38520.0..49031.99 ∨ 109504.0..142655.99 ∨
             214976.0..378687.99 ∨ 399680.0..409727.99 ∨
             420544.0..711679.99 ∨ 8237056.0..8347647.99
             ∨ 8392704.0..8790016.0]
             (t-weight: 18, u-weight: 10)
coyote  <::  [averageSystemTime = 0.07..46.22 ∨
             222.53..256.79 ∨ 402.67..418.92 ∨
             544.43..4579.19 ∨ 5448.53..25352.52] &
             [averageHogFactor = 0.85..2.66 ∨
             38520.0..49031.99 ∨ 109504.0..142655.99 ∨
             214976.0..378687.99 ∨ 399680.0..409727.99]
             (t-weight: 9, u-weight: 1)
```

Figure 40: Examples of GEM rules for coyote's computing behavior.

```
coyote  <::  [averageSystemTime <> 1414.4..3676.79 ∨
             4579.2..5448.52 ∨ 19399.47..447556.26] &
             [averageHogFactor = 0.0..0.84 ∨
             109504.0..142655.99 ∨ 214976.0..378687.99 ∨
             399680.0..409727.99 ∨ 420544.0..711679.99 ∨
             7000064.0..7196671.99 ∨
             8392704.0..8790016.0]
             (t-weight: 19, u-weight: 19)
```

Figure 41: Examples of AQ11 rules for coyote's computing behavior.

6.7 Partial Memory Incremental Learning using AQ11 and GEM

Once a set of representative examples has been computed, we can apply a variety learning approaches to the examples. We could apply a batch learning algorithm, as AQ-PM does, or apply an incremental learning algorithm. The final experimental comparison involves using the incremental learning algorithms of AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988) to learn concept descriptions from a partial memory of training examples. In other words, the learning function (step 6) in Algorithm 2 is performed by AQ11 and GEM. To distinguish these algorithms from when they are

executed in their normal modes, we will refer to them as AQ11-PM and GEM-PM, respectively.

To use AQ11 and GEM, which are integrated in AQ15 (Hong et al. 1986), with AQ-PM, for the initial learning step, AQ11 and GEM were used to form a set of rules from the first data partition. These rules and the first data partition were then passed to the AQ-PM system which computed a set of representative training examples. For subsequent learning steps, representative examples, past concept descriptions, and the set of new training examples are passed to AQ11 and GEM. Again, these programs produce sets of concept descriptions. These concept descriptions, the representative examples, and training examples are passed to the AQ-PM system which computes the new set of representative examples. The process repeats for the duration of the experiment. This experiment was managed by a script program that coordinated the data exchange between AQ-PM and AQ15.

Incremental learning is performed using the AQ11 no memory incremental learning algorithm (Michalski and Larson 1983) by inducing new concept descriptions from old concept descriptions, representative examples, and any new training examples. Conceptually, the method induces new rules (VL_1 complexes) from positive training examples that should be covered by the current concept description but are not. These new rules are subtracted from the rules comprising the negative concept description and added to the rules comprising the positive concept description. As pointed out by Michalski and Larson (1983), adding and subtracting rules is difficult and demonstrate how these operations can be performed as covering operations, performed by the AQ algorithm (Michalski 1969). See Michalski and Larson (1983) for more detail.

Incremental learning is performed using the GEM full memory incremental learning algorithm (Reinke and Michalski 1988) by inducing new concept descriptions from old concept descriptions, representative examples, and any new training examples.

Conceptually, the method first specializes those rules that cover any new negative examples. Once all rules have been specialized, they are generalized to cover any new positive events. Once this step has been performed, the rules will be complete and consistent with respect to the training examples accumulated thus far. See Reinke and Michalski (1988) for details.

Experiments were performed on the blasting caps and the reduced computer intrusion detection data sets. For these problems, the methods are compared to AQ-PM in terms of predictive accuracy, learning time, and concept complexity. Results for memory requirements are not presented, since the memory requirements for the algorithms are the same as those for AQ-PM in previous experiments using the respective data sets. For each partition, 50 learning and testing runs were performed.

6.7.1 Experimental Results for the Blasting Cap Data Set

The numerical results for the experimental comparison between AQ-PM, GEM-PM, and AQ11-PM for the blasting cap detection problem appear in Table 8. These numeric results are summarized graphically in Figures 42–44. Combining a partial memory model with AQ11 and GEM incremental learning (AQ11-PM and GEM-PM) did not produce more predictive rules than using AQ11 and GEM in their normal modes (i.e., no and full memory learning), comparing Figures 30 and 42. AQ11 and GEM achieved 100% and 99% predictive accuracy at partition 10, while AQ11-PM and GEM-PM achieved 95% and 97%, respectively. GEM-PM's performance was better in terms of predictive accuracy than AQ-PM's for this problem, as was AQ11-PM's, illustrated in Figure 42.

GEM-PM's learning times were better than AQ-PM's (Figure 43) and GEM's (comparing to Figure 31). AQ-PM and GEM learned 448% and 290% slower than GEM-PM (1.15s and 0.82s vs. 0.21s) This is a result of the fact that GEM-PM is learning from fewer examples than GEM and is not processing these examples to the same extent as AQ-

Table 8: Experimental results for the blasting cap detection problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)	Learning Times (sec)	Concept Complexity (rules/selectors)
1	AQ-PM	67.36±2.40	0.3	2.01/2.09
	GEM-PM	69.14±2.82	3.35	2.0/2.0
	AQ11-PM	67.12±2.63	3.35	2.0/2.04
2	AQ-PM	80.38±1.58	0.39	2.05/3.0
	GEM-PM	81.44±1.79	1.13	3.04/4.3
	AQ11-PM	80.78±1.98	1.42	2.4/3.18
3	AQ-PM	84.65±1.28	0.48	2.25/3.78
	GEM-PM	86.68±1.4	0.71	3.82/6.32
	AQ11-PM	86.02±1.41	0.81	3.06/4.5
4	AQ-PM	86.03±1.38	0.55	2.41/4.47
	GEM-PM	87.7±2.0	0.54	4.16/7.74
	AQ11-PM	89.74±1.02	0.47	3.68/6.08
5	AQ-PM	87.45±1.50	0.67	2.65/5.18
	GEM-PM	90.32±2.39	0.53	4.22/8.56
	AQ11-PM	91.72±1.0	0.56	4.18/7.6
6	AQ-PM	88.77±1.26	0.77	2.82/5.61
	GEM-PM	90.36±3.1	0.3	4.26/8.88
	AQ11-PM	93.1±0.87	0.48	4.64/8.8
7	AQ-PM	90.22±1.20	0.86	3.01/6.33
	GEM-PM	93.26±1.93	0.43	4.58/10.0
	AQ11-PM	94.08±0.89	0.5	5.3/10.56
8	AQ-PM	90.80±1.11	1.0	3.26/6.84
	GEM-PM	92.52±2.34	0.32	4.52/9.86
	AQ11-PM	95.22±0.78	0.46	5.72/11.92
9	AQ-PM	91.33±0.98	1.09	3.41/7.25
	GEM-PM	96.16±0.75	0.41	4.82/10.82
	AQ11-PM	96.1±0.84	0.5	6.06/12.84
10	AQ-PM	93.13±0.96	1.15	3.55/7.64
	GEM-PM	95.48±1.05	0.21	4.52/10.48
	AQ11-PM	96.84±0.68	0.5	6.46/14.06

PM because of incremental learning. GEM's concept descriptions (Figure 33) were 48% more complex than GEM-PM's (Figure 44), which in turn, were 37% more complex than AQ-PM's (15.56 vs. 10.48 vs. 7.64 selectors). Figure 45 shows the rules induced by GEM-PM for the blasting cap class.

AQ11's rules were 3% more predictive at partition 10 (Table 6) than AQ11-PM's (Table 8). AQ11-PM did show an improvement of 4% over AQ-PM learning (Figure 42). AQ11-PM learned more quickly than both AQ11 (Figure 31) and AQ-PM (Figure 43),

which is a result of providing the AQ11 incremental learning algorithm with fewer training examples. AQ11 learned 174% slower than AQ11-PM (1.37s vs. 0.5s), while AQ-PM learned 130% slower (1.15s vs. 0.5s). The most interesting of these results is the concept complexity results for AQ11-PM, shown in Figure 44. The previous experiment with AQ11 using the blasting caps data set showed fluctuations in concept complexity (Figure 33). This phenomenon is not present in this experiment for AQ11-PM, as shown in Figure 44. Apparently, the representative examples have dampened the effect of concept instability (O'Rourke 1982). Furthermore, the AQ11-PM concept descriptions are much simpler than the AQ11 concept descriptions. The AQ11-PM concept descriptions, however, are not as simple as the AQ-PM concept descriptions. AQ11 concept descriptions consisted of 56% more selectors than AQ11-PM (21.99 vs. 14.06), which in turn, consisted of 84% more selectors than AQ-PM (14.06 vs. 7.64). Figure 46 shows a decision rule induced by AQ11-PM for the caps class. Figure 21 shows similar rules induced by AQ-PM.

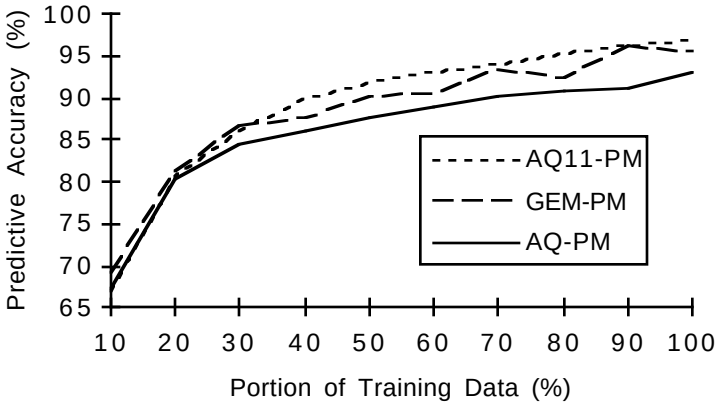


Figure 42: Predictive accuracy for the blasting cap detection problem.

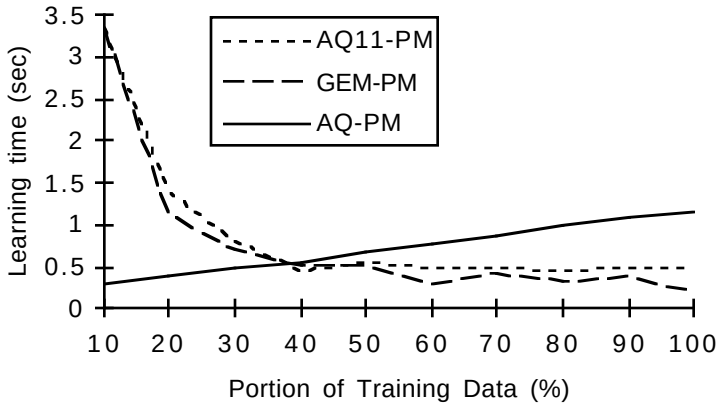


Figure 43: Learning times for the blasting cap detection problem.

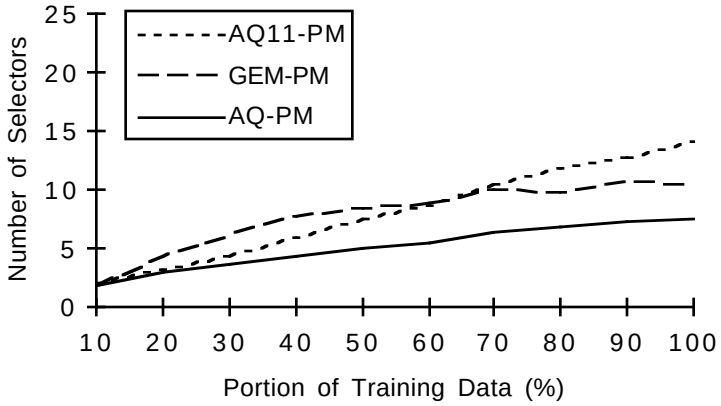


Figure 44: Concept complexity for the blasting cap detection problem.

```
cap  <::  [blobAverage = 8.00..24.99] &
          [blobCompactness2 = 4.71..18.45 ∨
          46.03..47.43]
          (t-weight: 5, u-weight: 3)
cap  <::  [rectangleWidth = 4.0..4.99]
          (t-weight: 5, u-weight: 1)
cap  <::  [blobMaximum = 61.00..131.99] &
          [blobAverage = 25.00..42.99]
          (t-weight: 5, u-weight: 1)
cap  <::  [blobLength = 5.00..6.99] &
          [rectangleWidth = 4.0..4.99 ∨ 5.0..7.99]
          (t-weight: 2, u-weight: 1)
```

Figure 45: Examples of GEM-PM rules for blasting caps.

```
cap  <::  [blobCompactness2 = 4.71..18.45 ∨
          24.86..31.67 ∨ 36.15..36.89 ∨ 40.32..43.72 ∨
          46.03..47.43 ∨ 58.6..87.77] &
          [rectangleIntensitySD >= 21.85..24.32]
          (t-weight: 6, u-weight: 6)
```

Figure 46: Example of an AQ11-PM rule for blasting caps.

6.7.2 Experimental Results for the Computer Intrusion Detection Problem

The numerical results for the experimental comparison between AQ-PM, GEM-PM, and AQ11-PM for the computer intrusion detection problem appear in Table 9. These numeric results are summarized graphically in Figures 47–49. The predictive accuracy results for the intrusion detection problem are similar to the blasting cap detection problem in that AQ11-PM and GEM-PM did produce slightly less predictive rules than AQ11 and GEM, comparing Figures 47 and 36. Both AQ11 and GEM achieved 100% on testing data at partition 10 (Figure 36), while AQ11-PM and GEM-PM achieved roughly 99% (Figure 47).

The difference in learning times between AQ11-PM and GEM-PM, and AQ-PM is greater for the intrusion detection problem (Figure 48) than it is for the blasting caps problem (Figure 43), although both graphs demonstrate the same trends. Using a partial memory model with AQ11 and GEM did produce better learning times than using these algorithms with their usual memory models, which can be seen by comparing Figures 37 and 48. Learning times for AQ-PM, and AQ11 and GEM without partial memory learning appear to increase as more and more training data is processed (Figure 37). Yet, learning times for AQ11-PM and GEM-PM actually decrease slightly as more training data is progressively received (Figure 48). AQ-PM learned 290% and 746% slower than AQ11-PM (10.91s vs. 2.8s) and GEM-PM (10.91s vs. 1.29s) at partition 10. On the other hand,

Table 9: Experimental results for the intrusion detection problem.

Partition	Learning System	Predictive Accuracy ± 95% CI (%)	Learning Times (sec)	Concept Complexity (rules/selectors)
1	AQ-PM	80.14±1.27	4.7	9.48/11.16
	GEM-PM	78.06±1.23	50.7	9.0/9.0
	AQ11-PM	78.7±1.11	48.18	9.0/9.0
2	AQ-PM	86.6±0.93	5.9	10.44/14.62
	GEM-PM	87.36±0.74	15.96	9.32/10.48
	AQ11-PM	86.58±0.8	16.1	9.26/10.08
3	AQ-PM	90.0±0.83	7.0	11.54/17.44
	GEM-PM	92.74±0.58	8.63	9.68/11.68
	AQ11-PM	91.8±0.67	9.6	9.68/11.62
4	AQ-PM	91.64±0.89	7.73	12.42/19.42
	GEM-PM	94.86±0.5	3.66	9.98/13.06
	AQ11-PM	95.0±0.49	5.32	10.18/13.16
5	AQ-PM	92.53±0.93	8.52	13.34/21.72
	GEM-PM	95.72±0.76	1.97	10.2/14.1
	AQ11-PM	96.22±0.46	3.48	10.58/14.84
6	AQ-PM	93.81±0.68	9.07	14.02/23.48
	GEM-PM	96.28±1.1	1.91	10.62/15.72
	AQ11-PM	97.6±0.31	3.31	11.36/17.62
7	AQ-PM	94.75±0.76	9.55	14.4/24.88
	GEM-PM	97.3±1.0	1.5	11.04/17.36
	AQ11-PM	98.2±0.31	2.65	11.94/19.5
8	AQ-PM	96.19±0.71	9.85	14.88/25.52
	GEM-PM	97.48±1.14	1.39	11.4/18.84
	AQ11-PM	98.76±0.25	2.68	12.36/21.04
9	AQ-PM	96.45±0.62	10.41	15.52/27.22
	GEM-PM	98.08±1.13	1.48	11.92/20.76
	AQ11-PM	99.58±0.17	2.9	12.84/22.7
10	AQ-PM	97.14±0.59	10.91	15.96/28.22
	GEM-PM	99.26±0.61	1.29	12.22/22.04
	AQ11-PM	99.8±0.13	2.8	13.42/24.56

AQ11 learned 283% slower than AQ11-PM (10.72s vs. 2.8s), while GEM learned 485% slower than GEM-PM (7.55s vs. 1.29s) at partition 10.

Finally, comparing concept complexity, the incrementally learned concept descriptions (AQ11-PM and GEM-PM) were slightly simpler than the temporally batch learned concept descriptions (AQ-PM) for the intrusion detection problem (Figure 49). At partition 10, AQ-PM's rules consisted of 15% more selectors than AQ11-PM's (28.22 vs.

24.56) and 28% more than GEM-PM's (28.22 vs. 22.04). The concept complexity results for AQ11-PM and GEM-PM were also simpler compared to AQ11 and GEM. AQ11's rules consisted of 25% more selectors than AQ11-PM (30.8 vs. 24.56), while GEM's rules consisted of 14% more (25.1 vs. 22.04). For the blasting caps detection problem, however, recall that AQ-PM produced simpler descriptions than AQ11, GEM, AQ11-PM, and GEM-PM (Figures 33 and 39). For these experiments, as with the experiments using the blasting caps data set, AQ11-PM shows no signs of concept instability. Again, it appears that the representative examples have stabilized fluctuations in concept instability for AQ11. Figures 50 and 51 show rules for coyote's computing behavior induced at partition 10 by GEM-PM and AQ11-PM. Figure 9 shows rules for the same class induced by AQ-PM.

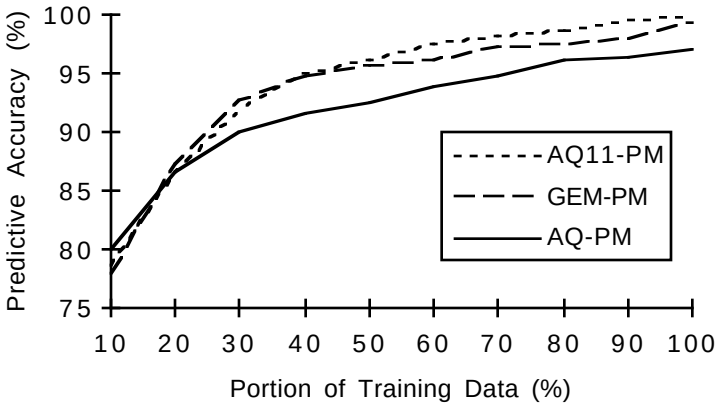


Figure 47: Predictive accuracy for the intrusion detection problem.

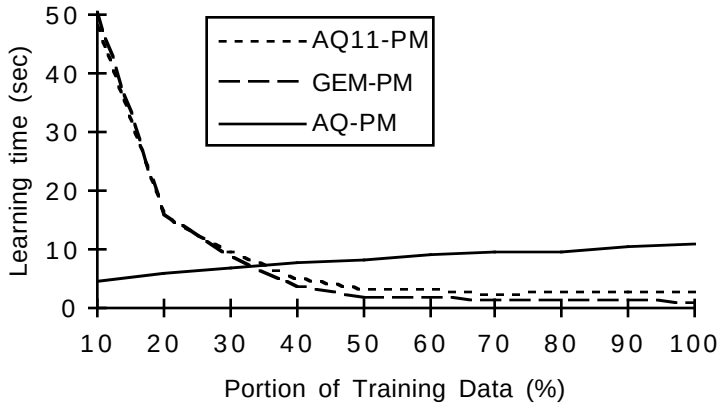


Figure 48: Learning times for the intrusion detection problem.

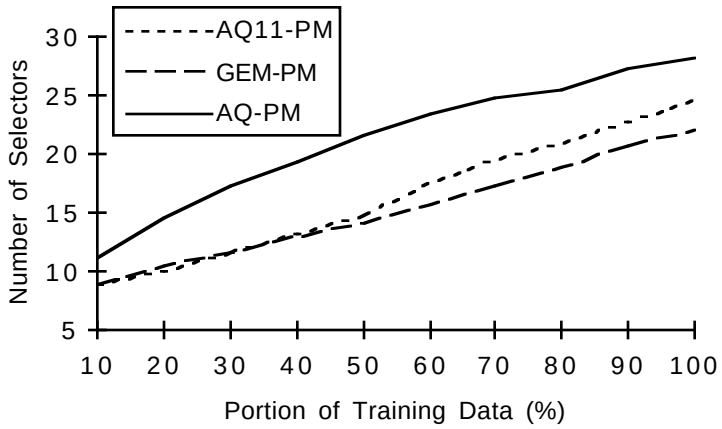


Figure 49: Concept complexity for the intrusion detection problem.

```
coyote  <::  [averageBlocks = 0.23..0.32 ∨ 0.43..0.47 ∨
             0.48..0.56 ∨ 1.32..1.37 ∨ 36.11..124.0] &
             [averageHogFactor <> 6.57..24807.99 ∨
             38520.0..96831.99]
             (t-weight: 5, u-weight: 5)
```

Figure 50: Examples of GEM-PM rules for coyote's computing behavior.

```
coyote  <::  [averageSystemTime <> 1414.4..3676.79 ∨
             193399.47..447556.26 ∨ 691404.8..691404.8] &
             [averageHogFactor = 0.0..0.84 ∨
             96832.0..109503.99 ∨ 214976.0..378687.99 ∨
             399680.0..409727.99 ∨ 420544.0..711679.99 ∨
             8237056.0..8347647.99 ∨
             8392704.0..8790016.0]
             (t-weight: 7, u-weight: 7)
```

Figure 51: Example of an AQ11-PM rule for coyote's computing behavior.

6.8 Summary of Experimental Results

The progressive partial memory learning methodology was validated against several learning algorithms (i.e., AQ-BPL, AQ11, GEM, and FLORA) using a variety of problems (i.e., computer intrusion detection, blasting cap detection, email sorting, and the STAGGER concepts). For the problems considered, AQ-PM, compared to AQ-BPL, considerably reduced memory requirements and learning time, but produced slightly less predictive concept descriptions. On average, AQ-BPL produced concept descriptions that were 4.6% more predictive than AQ-PM concept descriptions, but AQ-BPL was 77% slower and maintained 236% more examples than AQ-PM. AQ-PM, compared to FLORA2 (Widmer and Kubat 1996), comparably tracked changing concepts, but used considerably less memory. AQ-PM, compared to AQ11 (Michalski and Larson 1983) and GEM (Reinke and Michalski 1988), produced slightly lower predictive accuracy, lower memory requirements than GEM, but higher memory requirements than AQ11, comparable learning times, and comparable or simpler concept descriptions. Using a partial memory approach with AQ11 and GEM incremental learning was beneficial. Comparing to AQ11 and GEM, although predictive accuracy was slightly decreased, learning times were improved and concepts were more stable and simpler.

Chapter 7: Discussion

7.1 On Parameter Settings

As with most learning systems, AQ-PM's ability to properly learn and track concepts is dependent on its parameter settings. For each of the problems discussed, excluding the aging, forgetting, and inductive support parameters, all possible settings for the AQ-PM learning parameters were investigated. After determining the best set of parameters, the remaining parameters were investigated to determine if an increase in performance resulted. These additional parameters included the forgetting parameter and the criterion parameters. The primary objective of the experimentation was to thoroughly understand the partial memory mechanisms of the learning system.

Experimentation showed that of the AQ-PM learning parameters, the intersection parameter produced the greatest effect on learning performance. As shown in the next section, Section 7.2, the borders intersection method generally produced higher predictive accuracy results. This is directly related to the increased number of representative examples maintained — a result of selecting examples that lie on the borders of concept descriptions. It follows that maintaining more representative examples results in longer learning times.

The `update` and `useinhypos` parameters should be set on the basis of the stability of a problem. For instance, if the `update` parameter were set to accumulate (`accum`) all representative examples and the `useinhypos` parameter were set to use previously induced hypotheses (`yes`), then the partial memory learner would use a lot of past knowledge. The more past knowledge used for learning, the more anchored the learner will be with respect to the past. In other words, the learner will become less reactive.

Conversely, if the learner were set to re-evaluate all representative examples (`update` set to `reeval`), disregard past hypotheses (`useinhypos` set to `no`), prefer rules based on new examples (criterion set to `minold`), and forget examples older than some age, then the partial memory learner will be very reactive. This reactivity can sometimes lead to undesirable results, such as catastrophic forgetting (see Section 7.3).

Unfortunately, experimental results for the problems considered do not bear out this intuition. This is not to say, however, that the intuition is incorrect. On the STAGGER problem, a problem that intuitively demanded reactivity from the learner, the `reeval` `update` parameter did not produce better results than the `accum` `update` parameter. During the first time period (i.e., time steps 1–40), the learner using the `reeval` parameter did converge more quickly to the target concept than it did using the `accum` parameter, but after the concept change at time step 40, the system could never recover to an acceptable level of performance.

In general, the update parameters will perform best (i.e., keep the right representative examples) on concepts that are either static or expanding (i.e., increasing in size or area). In this case, the set of representative examples will either remain the same, or will be continually refreshed by those new training examples that provide evidence of the expanding concept. On the other hand, if the concept is moving in the space (i.e., changing location, but not necessarily changing size or shape), then the current methods of representative example selection have difficulty removing the examples that lie on the trailing edge of the concept. Since these examples appear on the trailing edge of the concept, then the selection methods will always consider them representative and recycle them back into the set of representative examples. Since they are present in the set of representative examples, learning will produce a concept with these examples lying on its border. This process will continually repeat until these examples producing the stasis are

removed. This is why explicit forgetting mechanisms are required: to explicitly remove those representative examples that are likely to lie on the trailing edge of moving concepts.

Needing explicit forgetting techniques can be both an advantage and a disadvantage depending on the type of changing concept the system needs to learn. Assume that a concept is expanding along a single dimension in the representation space. If training examples on the trailing edge of the concept (i.e., on the dimensions that are not expanding) do not reoccur, then memory- or a time-based forgetting techniques will eventually remove the representative examples that establish the concept boundary resulting in the learning of the wrong concept. Ultimately, performance will suffer.

Conversely, if the concept is moving in the representation space and forgetting mechanisms are not active, then the system will view the concept as if it is an expanding concept. A potentially interesting direction for further study would be to develop a taxonomy of changing concepts (i.e., moving, expanding, shrinking, and the like) and investigate methods for determining the types of concept change present during learning. Such methods would provide techniques for adaptive parameter tuning.

7.2 Corners versus Borders

Experimentation revealed that of the AQ-PM parameters the intersection parameter produced the greatest affect on learning performance. Two of the three identified methods for selecting representative examples were implemented: the corners method and the borders method. The surfaces method was not. For the problems considered, the borders method generally produced the highest predictive accuracies, although the corners method resulted in lower memory requirements, and hence, faster learning times. Performance comparisons between the borders method and the corners method for the blasting cap detection problem are shown in Figures 52–55. The curves for the email learning agent

and STAGGER problems were similar. The difference in performance between the borders and corners method for the computer intrusion detection problem was negligible.

It is anticipated, based on the findings presented, that the surfaces method, when implemented, would produce slightly higher predictive accuracies, higher memory requirements, and longer learning times than the borders method. Concept complexity is also likely to increase.

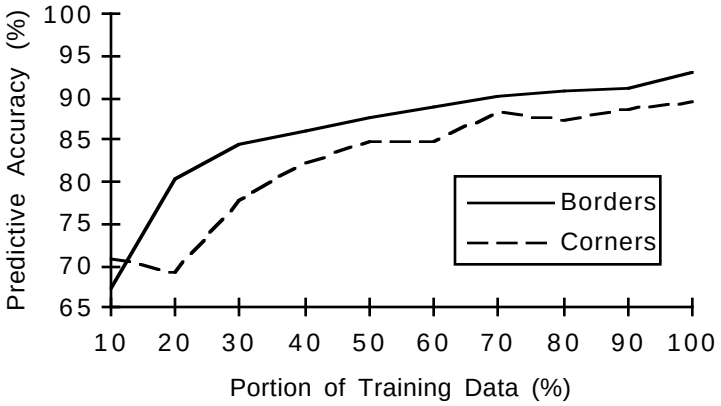


Figure 52: Borders versus corners comparison using predictive accuracy for the blasting cap detection problem.

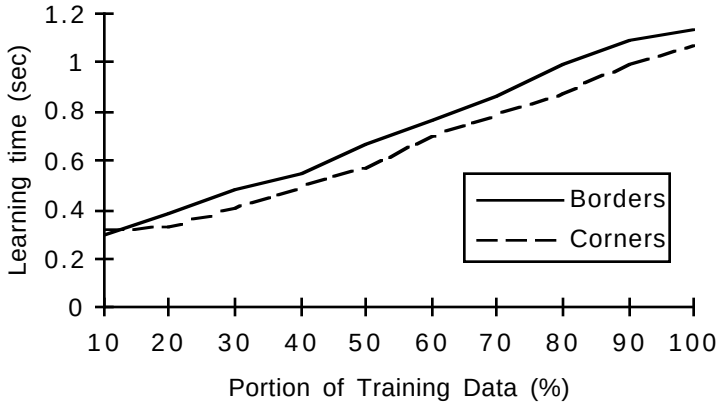


Figure 53: Borders versus corners comparison using learning times for the blasting cap detection problem.

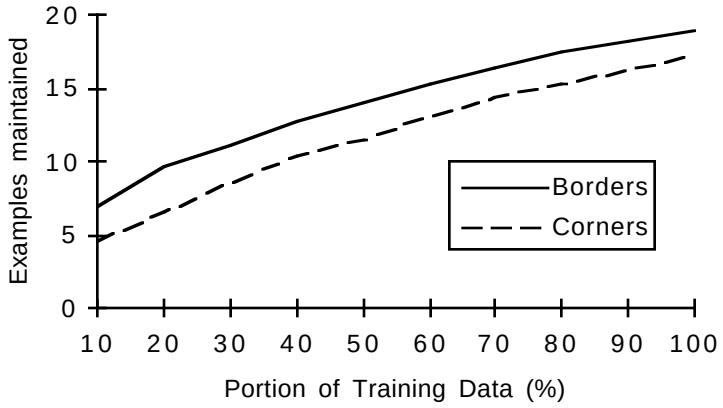


Figure 54: Borders versus corners comparison using memory requirements for the blasting cap detection problem.

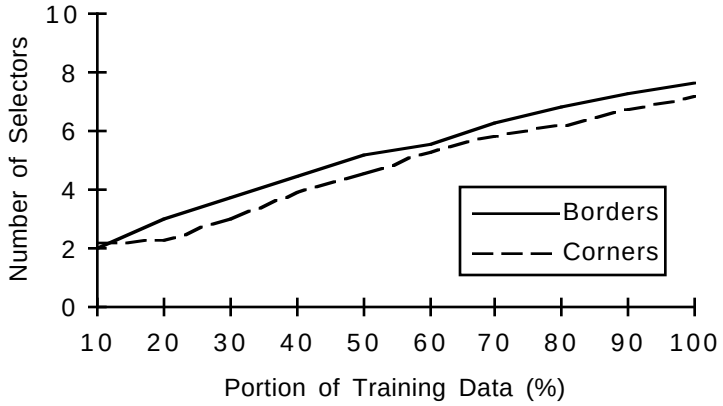


Figure 55: Borders versus corners comparison using concept complexity for the blasting cap detection problem.

7.3 Catastrophic Forgetting

What happens when the system forgets all of its representative examples? Problems like the email learning agent problem for which we assumed that more than one training example is presented to the learner, such a situation rarely arises, since there is always enough data contained in a partition. For the STAGGER problem, however, such a problem is brought on by a variety of situations.

The representation space for the STAGGER problem is small: consisting of three attributes, each with three values. Since training examples are randomly drawn from this space, over the course of a 120 time step experiment, any given training example is likely to be generated numerous times. Further, since the concepts are changing, it is also likely that before a concept shift, a training example will belong to the positive class, and after the concept shift, the same training example will belong to the negative class. For weight learning systems, such as STAGGER or Winnow, such an event is hardly catastrophic since both systems simply adjust connection weights accordingly, but for partial memory systems, in which explicit training examples are stored in memory and used for learning, such an event causes problems.

During experimentation with the STAGGER concepts, after a concept shift, there were situations in which there were two representative examples in a class. Assume the class is positive. A new negative training example arrives that is identical to one of the positive representative examples. We want to learn from this new negative training example. If the `ambig` parameter is set to prefer negative examples, then the situation is handled correctly. That is, one of the positive representative examples is discarded. Had the `ambig` parameter been set to prefer positive examples, then the new training example would have been discarded. The problem is that we can never know a priori what the correct `ambig` parameter is, since it will sometimes need to be positive and sometimes need to be negative. Setting the `ambig` parameter to empty, so both conflicting examples are removed is also a problem, since for that time step, the system learns nothing.

Further assume that in addition to the above scenario, that the other positive representative example is identical to an existing negative representative example. If the `ambig` parameter is set to negative, then all positive representative examples will be discarded. Catastrophic forgetting has occurred. While the above description sounds unlikely, it actually occurred frequently during experimentation.

This was the impetus behind the `recent ambig` parameter. The solution was to develop a version of the `ambig` parameter that is time dependent, rather than class dependent. In other words, an `ambig` parameter was needed that gives preference to the newest example, whether a training example or a representative example, irrespective of its class label.

7.4 On Concept Complexity

One surprising experimental result is that the concept complexity of the AQ-PM concepts is equal to or lower than the concept complexity of the AQ-BPL concepts. Initial experiments with the methodology, which were not fully automated, yielded concepts that were more complex (Maloof and Michalski 1995a, b). Reinke and Michalski (1988) also observed that full memory incremental learning produced more complex concept descriptions than batch learning. As mentioned, O'Rourke (1982) noticed wide variations in concept complexity when using no memory incremental learning. Figure 56 shows how AQ11-induced concepts fluctuate wildly from one learning run to the next. At the 7th partition, when an additional 10% of the data is presented to AQ11, which represents about 6 training examples, the concept complexity jumps from 13 selectors to 108. At partition 9, the number of selectors returns to a more acceptable level of 27. These radical jumps in the number of rules and the number of selectors occurred frequently.

The expectation at the beginning of this study, based on past research, was that partial memory learning, when implemented, would also yield more complex concept descriptions. The experimental results presented show that concept complexity is comparable to the batch learned concepts, as is the case with the intrusion detection and email learning agent problem, or is less complex than the batch learned concepts, as is the case with the blasting cap detection problem. Although for the blasting cap problem, AQ-PM's predictive accuracy was not as close to its batch learning counterpart as it was on

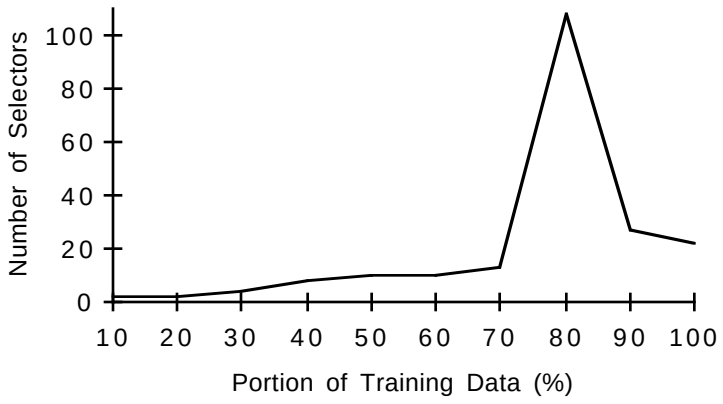


Figure 56: An example of concept instability from an AQ11 experiment.

other problems, namely the email learning agent and computer intrusion detection problems. AQ-PM's concept descriptions were much less complex than those for both AQ11 and GEM. Combining a partial memory approach with the incremental learning methods of AQ11 and GEM produced simpler concept descriptions than these methods produced with their usual memory models. Further, combining AQ11 incremental learning with a partial memory model (AQ11-PM) eliminated much, if not all, of the concept instability (O'Rourke 1982) observed in the experiments involving AQ11 no memory incremental learning.

In an ideal situation, such as that shown using the Iris data set (Fisher 1936) in Section 4.4, the concept complexity of a discriminant concept description induced from all training examples will have the same concept complexity of a discriminant concept description induced from the representative examples. Note that this statement is not true for characteristic concept descriptions.

Unfortunately, real-world representation spaces are not clean and well-defined, so we cannot expect that for all cases, the concept complexity of concept descriptions induced

from representative examples will be the same as that of concept descriptions induced from all training examples, especially when learning is conducted over time. What this notion gives us, however, is a heuristic measure of the success of the representative example selection process. We can expect that when the concept complexity of AQ-PM-induced concept descriptions is close to the complexity of batch-learned concept descriptions, that the system is selecting the correct representative examples. If, on the other hand, concept complexity of AQ-PM-induced concept descriptions is higher or lower than batch-learned concept descriptions, it may be in indication that the representative examples are not really capturing the essence of the concepts. This would explain why the difference in predictive accuracy was greatest on the problem in which the difference in concept complexity was the greatest, and why the difference in predictive accuracy was lowest on problems in which the difference in concept complexity was the lowest. Experimentation beyond that presented here is needed to confirm or deny this conjecture.

Chapter 8: Conclusions

In this dissertation, a new learning methodology based on the idea of maintaining a partial memory of past examples was presented. Committing to maintaining some of the past training examples entails discussing mechanisms to select those examples. Learning naturally occurs over time, so aging mechanisms appear useful. Over time, the system is likely to make several observations of the same phenomenon, so a logical question is, What can inductive support mechanisms contribute? If the system operates under the assumption that concepts may change or drift, then the system needs mechanisms to remove useless or out-dated knowledge, such as forgetting processes.

Key components of the methodology were implemented in an experimental system so these ideas could be investigated and validated. Experimental results on a variety of different problems using various learning algorithms soundly demonstrated that these mechanisms for selecting, maintaining, and forgetting training examples are important. Real-world applications, such as intelligent agents, dynamic knowledge-bases, and computer vision systems, require fast learning systems that operate with low memory requirements and most importantly, cope with concept change. Finally, if humans expect to use machine learning algorithms to better understand the problems to which they are applied, simple, comprehensible concept descriptions are also necessary.

8.1 Contributions

The key contributions of the work presented in this dissertation to the field of machine learning are as follows:

- development of novel inductive learning methodology
- development of novel schemes for selecting representative examples
- implementation of an experimental system (AQ-PM) based on the methodology
- statistically and methodologically sound validation of the experimental system
- application of the experimental system to a variety of relevant real-world problems
- demonstration that the method produces considerable improvements in memory requirements and learning time with slight decreases in predictive accuracy
- demonstration that the method can be used across a variety of learning methods
- comparison to and considerable improvement over existing progressive learning methods and methods for tracking concept drift

8.2 Weaknesses

Aside from the weaknesses stemming from the assumptions made in Section 2.1 (e.g., static representation space, stationary contexts, representational assumptions, and the like), weaknesses exist in the methodology, the AQ-PM implementation, and the experimentation. With regards to the methodology, a primary weakness is that the methodology is vague with respect to periods of time in which there is no user feedback. Realistically, we cannot expect that systems will always have the benefit of feedback. A logical assumption is that no feedback implies correct behavior on the part of the system, which is not necessarily problematic. The difficulty arises when feedback does occur and the system realizes that many of its past decisions which were not accompanied with feedback were wrong. The simplest approach is to simply do nothing and forget the past. There may be applications in which this memoryless approach is unacceptable and more

sophisticated schemes will be required to backtrack, review, and reprocess past observations.

A related problem is when feedback is available for a period of time, but at some point, the system realizes that the feedback for this period was wrong. We have a similar problem as above, except that phenomena were accompanied by feedback. Again, the methodology is vague with respect to this point in that we assumed that feedback was correct. Future work will allow us to relax this assumption.

With regards to the implementation of the methodology, AQ-PM, small portions of the methodology were not implemented. These small portions include the surfaces representative example selection, which was actually not conceived as a selection method until late in this study, memory-based forgetting, since this is a well-studied form of forgetting, and frequency-based forgetting, since, for the problems considered, experimental results did not firmly establish the necessity for inductive support mechanisms.

With regards to the experimentation of the AQ-PM system, broader and deeper experimentation is needed. The objective of the experimentation was to show that the ideas presented in this dissertation are valuable and deserve continued investigation. Hopefully this objective was achieved. If so, then AQ-PM needs to be tested on additional problems, real and synthetic, to better understand its abilities and weaknesses. Furthermore, the value of various parameters of the system needs to be better explored. For example, it would be interesting to investigate the effect of various aging functions for a class of problems. More direct comparisons to other systems would also be worthy of study.

8.3 Future Work

Future work in this area can take place on many different levels. At the methodological level, weakening the assumptions made and investigating how, for example, constructive

induction applies in this context. For a given problem, perhaps constructive induction is not a good idea for a certain period of time, but an event occurs, such as feedback or concept change, that necessitates modifying the representation space. The methodology also has an implicit assumption that feedback was always available. An interesting area of inquiry would be to better understand how a progressive learning system copes with periods without the benefit of feedback and how it should behave once feedback arrives, especially if that feedback contradicts past decisions.

One of the objectives of this work was to build a research platform so that ideas for representative example selection, aging mechanisms, and forgetting mechanisms could be explored and tested. Even so, a few elements of the methodology remain unimplemented, such as the surfaces method for finding representatives examples and frequency-based forgetting. The implementation of the surfaces selection method would provide additional opportunities for experimentation and comparison with the corners and borders methods.

Incorporating additional ideas from other researchers would also be fruitful. Noise was not explicitly handled, but the mechanisms for handling noise proposed by Aha et al. (1992) and used by Widmer and Kubat (1996) could be easily incorporated. Furthermore, Widmer and Kubat's (1996) notion of adaptive forgetting schemes could be used. Currently, the window size for time-based forgetting is static. The heuristics used by the FLORA systems could be adapted and incorporated into AQ-PM.

With regards to experimental data, it would be useful if a real-world data set could be generated in which the characteristics of changing concepts could be documented and understood. We can intuitively think about a problem, such as the email learning agent, and rationalize that, yes, there is the possibility of concept change. We can collect data for the problem, but for a sufficiently complex problem, it is difficult to analyze the data and precisely understand what types of change are occurring and the times at which that change occurs. The machine learning research community would benefit greatly from a data set

collected over a period when concept change was highly probable, like the period at the end of one semester and the beginning of the next.

Another interesting area of application of the progressive partial memory learning methodology, proposed by Wnek (1996), is to the problem of learning with large data sets. In some sense, we have come full circle, since much of the early work in incremental learning expressly dealt with this problem, although it was not an explicit target of this study. PPML could be applied to this problem by splitting a large data set into several small partitions and running AQ-PM on each partition. It is not clear, however, especially from the experimental results presented, if the total time required for AQ-PM to learn from each partition would be less than the time required for AQ15c to learn from the entire data set. Of course this point becomes moot if the entire data set is too large to load into memory. A partial memory approach, especially if combined with an incremental learning method like AQ11, becomes an obvious alternative.

Bibliography

Aha, D.W.; Kibler, D.; and, Albert, M.K. (1991) Instance-based learning algorithms. *Machine Learning* 6:37–66.

Aloimonos, Y.; Weiss, I.; and Bandopadhay, A. (1988) Active vision. *International Journal of Computer Vision* 7:333–356.

Bala, J.W. (1992) Incremental learning of a large number of noisy texture classes by similarity measure. *Proceedings of the International Conference on Image Processing and its Applications*, 594–597.

Becker, J.M. (1985) Topics in incremental learning of discriminant descriptions. Technical Report UIUCDCS-F-85-935, Department of Computer Science, University of Illinois, Urbana, IL.

Baim, P.W. (1988) A method for attribute selection in inductive learning system. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.6:888–896.

Ballard, D., and Brown, C. (1993) Principles of animate vision. In Aloimonos, Y., ed., *Active Perception*. 245–282. Hillsdale, NJ: Lawrence Erlbaum Associates.

Bhandaru, M.K., and Murty, M.N. (1991) Incremental learning from examples using HC-expressions. *Pattern Recognition* 24.4:273–282.

Blake, A.; Isard, M.; and Reynard, D. (1995) Learning to track the visual motion of contours. *Artificial Intelligence* 78:179–192.

Bloedorn, E.; Wnek, J.; Michalski, R.S.; and Kaufman, K. (1993) AQ17 — A multistrategy learning system: the method and user's guide. *Reports of the Machine Learning and Inference Laboratory*, MLI 93–12. Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.

Bloedorn, E.; Mani, I.; and MacMillan, T.R. (1996) Machine learning of user profiles: representational issues. *AAAI-96*, 433–438..

Bloedorn, E., and Michalski, R.S. (1996) ELA: the email learning agent. *Reports of the Machine Learning and Inference Laboratory*, MLI 96–7. Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.

Blum, A. (1995) Empirical support for Winnow and Weighted-Majority based algorithms: results on a calendar scheduling domain. *Proceedings of the Twelfth International Conference on Machine Learning*, 64–72.

Chen, H.; Houston, A.; Nunamaker, J.; and Yen, J. (1996) Toward intelligent meeting agents. *IEEE Computer* 29.8:62–70.

Clyde, S.; Zhang, J.; and Yao, C. (1995) An object-oriented implementation of adaptive classification of job openings. *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, 9–16.

Cobb, H.G., and Grefenstette, J.J. (1993) Genetic algorithms for tracking changing environments. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 523–530.

Cohen, L.J. (1977) *The probable and provable*. Oxford: Oxford University Press.

Cohen, L.J. (1989) *An introduction to the philosophy of induction and probability*. Oxford: Clarendon Press.

Cohen, W.W. (1994) Incremental abductive EBL. *Machine Learning* 15:5–24.

Dasgupta, D., and McGregor, D.R. (1992) Nonstationary function optimization using the structured genetic algorithm. In Männer, R., and Manderick, B., eds., *Parallel Problem Solving from Nature*, 2:145–151. The Netherlands: Elsevier Science.

Davis, J. H. (1981) CONVART: a program for constructive induction on time dependent data. Master's Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Denning, D.E. (1987) An intrusion-detection model. *IEEE Transactions on Software Engineering* SE-13.2 (February) 222–232.

Duric, Z.; Fayman, J.A.; and Rivlin, E. (1996) Function from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.6:579–591.

Feigenbaum, E.A. (1963) The simulation of verbal learning behavior. In Feigenbaum, E.A., and Feldman, J., eds., *Computers and Thought*, 297–309. McGraw-Hill: New York, NY.

Fisher, R. (1936) The use of multiple measurements in taxonomic problems. *Annuals of Eugenics* 7:179–188.

Fisher, D. (1987) Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

Freivalds, R.; Kinber, E.; and Smith, S.H. (1995) On the impact of forgetting on learning machines. *Journal of the ACM* 42.6:1146–1168.

Frisch, Æ (1991) *Essential system administration*. Sebastopol, CA: O'Reilly & Associates.

Gennari, J.H.; Langley, P.; and Fisher, D. (1989) Models of incremental concept formation. In Carbonell, J.G., ed., *Machine Learning: Paradigms and Methods*. 11–61. Elsevier Science: Amsterdam.

Grefenstette, J.J. (1992) Genetic algorithms for changing environments. In Männer, R., and Manderick, B., eds., *Parallel Problem Solving from Nature*, 2:137–142. The Netherlands: Elsevier Science.

Gross, K.P. (1988) Incremental multiple concept learning using experiments. *Proceedings of the Fifth International Conference on Machine Learning*, 65–72.

Hirsh, H. (1994) Generalizing version spaces. *Machine Learning* 17:5–46.

Hong, J.; Mozetic, I.; and Michalski, R.S. (1986) AQ15: incremental learning of attribute-based descriptions from examples, the method and user's guide. Technical Report UIUCDCS-F-86-949. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Iba, W.; Woogulis, J.; and Langley, P. (1988) Trading simplicity and coverage in incremental concept learning. *Proceedings of the Fifth International Conference on Machine Learning*, 73–79.

Keppel, G.; Saufley, W.H.; and Tokunaga, H. (1992) *Introduction to design and analysis*, 2nd ed. New York, NY: W.H. Freeman.

Kerber, R. (1992) ChiMerge: discretization of numeric attributes. *AAAI-92*. 123–128.

Kibler, D., and Aha, D.W. (1987) Learning representative exemplars of concepts: an initial case study. *Proceedings of the Fourth International Workshop on Machine Learning*, 24–30.

Krizakova, I., and Kubat, M. (1992) FAVORIT: concept formation with aging of knowledge. *Pattern Recognition Letters* 13:19–25.

Kubat, M., and Krizakova, I. (1992) Forgetting and aging of knowledge in concept formation. *Applied Artificial Intelligence* 6:195–206.

Lebowitz, M. (1986) Concept learning in a rich input domain: generalization-based memory. In: Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., eds., *Machine Learning: An Artificial Intelligence Approach*, Vol. II, 193–214. Tioga: Palo Alto, CA.

Lee, W.D., and Ray, S.R. (1986) Rule refinement using the probabilistic rule generator. *AAAI-86*, 442–447.

Lim, J.; Lui, H.; and Wang, P. (1992) A framework for integrating fault diagnosis and incremental knowledge acquisition in connectionist expert systems. *AAAI-92*, 159–164.

Littlestone, N. (1989) Mistake bounds and logarithmic linear-threshold learning algorithms. Ph.D. Dissertation, University of California, Santa Cruz.

Littlestone, N. (1991) Redundant noisy attributes, attribute errors, and linear-threshold learning using Winnow. *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, 147–156.

Littlestone, N., and Warmuth, M.K. (1994) The Weighted Majority Algorithm. *Information and Computation* 108.2 (February) 212–261.

Maes, P. (1994) Agents that reduce work and information overload. *Communications of the ACM* 37.7 (July) 31–40.

Maloof, M.A., and Michalski, R.S. (1995a) A partial memory incremental learning methodology and its application to intrusion detection. *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, 392–397.

Maloof, M.A., and Michalski, R.S. (1995b) A partial memory incremental learning methodology and its application to intrusion detection. *Reports of the Machine Learning and Inference Laboratory*, MLI 95–2. Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA.

Maloof, M.A., and Michalski, R.S. (1995c) Learning evolving concepts using a partial memory approach. *Working Notes of the 1995 AAAI Fall Symposium on Active Learning*, 70–73.

Maloof, M.A., and Michalski, R.S. (1995d) Learning symbolic descriptions of 2D shapes for object recognition in x-ray images. *Proceedings of the 8th International Symposium on Artificial Intelligence*, 124–131.

Maloof, M.A., and Michalski, R.S. (1996) Learning symbolic descriptions of shape for object recognition in x-ray images. *Expert Systems with Applications*, (in press).

Maloof, M.A.; Duric, Z.; Michalski, R.S.; and Rosenfeld, A. (1996) Recognizing blasting caps in x-ray images. *Proceedings of the 1996 Image Understanding Workshop*, 1257–1261.

Markovitch, S., and Scott, P.D. (1988) The role of forgetting in learning. *Proceedings of the Fifth International Conference on Machine Learning*, 459–465.

Mehler, G.; Bentrup, J.; and Riedesel, J. (1986) INDUCE 4: a program for incrementally learning structural descriptions from examples. Technical Report, Department of Computer Science, University of Illinois, Urbana, IL.

Merz, C.J., and Murphy, P.M. (1996) UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Department of Information and Computer Science, University of California, Irvine.

Michalski, R.S. (1969) On the quasi-minimal solution of the general covering problem. *Fifth International Symposium on Information Processing*, A3:125–128.

Michalski, R.S. (1973) AQVAL/1 — computer implementation of a variable-valued logic system VL_1 and examples of its application to pattern recognition. *First International Joint Conference on Pattern Recognition*, 3–17.

Michalski, R.S. (1980) Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.4:349–361.

Michalski, R.S. (1983) A theory and methodology of inductive learning. In Michalski, R.S.; Carbonell, J.; and Mitchell, T.M., eds., *Machine Learning: An Artificial Intelligence Approach*, Vol. I, 83–134. Los Altos, CA: Morgan Kaufmann.

Michalski, R.S. (1985) Knowledge repair mechanisms. *Proceedings of the Third International Machine Learning Workshop*, 116–119.

Michalski, R.S. (1994) Inferential Theory of Learning: developing foundations for multistrategy learning. In Michalski, R.S., and Tecuci, G., eds., *Machine Learning: A Multistrategy Approach*, Vol. IV, 3–61. San Francisco, CA: Morgan Kaufmann.

Michalski, R.S. (1996) Personal communication.

Michalski, R.S., and Chilausky, R.L. (1980) Learning by being told and learning from examples: an experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems* 4.2:125–161.

Michalski, R.S., and Larson, J.B. (1978) Selection of most representative training examples and incremental generation of VL_1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11. Technical Report 867, Department of Computer Science, University of Illinois, Urbana, IL.

Michalski, R.S., and Larson, J.B. (1983) Incremental generation of VL_1 hypotheses: the underlying methodology and the description of program AQ11. Technical Report UIUCDCS-F-83-905, Department of Computer Science, University of Illinois, Urbana, IL.

Mitchell, T.; Caruana, R.; Freitag, D.; McDermott, J.; and Zabowski, D. (1994) Experience with a Learning Personal Assistant. *Communications of the ACM* 37.7 (July) 81–91.

Nevins, A.J. (1995) A branch and bound incremental conceptual clusterer. *Machine Learning* 18:15–22.

O'Rorke, P. (1982) A comparative study of inductive learning systems AQ11 and ID3. *Intelligent Systems Group Report* 82-2. Department of Computer Science, University of Illinois, Urbana.

Pitas, I.; Milios, E.; and Venetsanopoulos, A.N. (1992) A minimum entropy approach to rule learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics* 22.4 (July/August) 621–635.

Reinke, R.E. (1984) Knowledge acquisition and refinement tools for the ADVISE META-EXPERT system. Master's Thesis. University of Illinois, Urbana, IL.

Reinke, R.E., and Michalski, R.S. (1988) Incremental learning of concept descriptions: a method and experimental results. In Hayes, J.E.; Michie, D.; and Richards, J., eds., *Machine Intelligence 11*, 263–288. Oxford: Clarendon Press.

Rumelhart, D.E.; Hinton, G.E.; and Williams, R.J. (1986) Learning internal representations by error propagation. In Rumelhart, D.E., and McClelland, J.L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, 318–362. Cambridge MA: MIT Press.

Sablon, G.; De Raedt, L.; and Bruynooghe, M. (1994) Iterative version spaces. *Artificial Intelligence* 69:393–409.

Sablon, G., and De Raedt, L. (1995) Forgetting and compacting data in concept learning. *IJCAI-95*, 432–438.

Salganicoff, M. (1993) Density-adaptive learning and forgetting. *Proceedings of the Tenth International Conference on Machine Learning*, 276–283.

Salganicoff, M.; Ungar, L.H.; and Bajcsy, R. (1996) Active learning for vision-based robot grasping. *Machine Learning* 23:251–278.

Schlimmer, J.C., and Fisher, D. (1986) A case study of incremental concept induction. *AAAI-86*, 496–501.

Schlimmer, J.C., and Granger, R.H. (1986) Beyond incremental processing: tracking concept drift. *AAAI-86*, 502–507.

Schlimmer, J.C. (1987a) Concept acquisition through representational adjustment. Ph.D. Dissertation, University of California, Irvine.

Schlimmer, J.C. (1987b) Incremental adjustment of representations for learning. *Proceedings of the Fourth International Workshop on Machine Learning*, 79–90.

Schum, D.A. (1994) *The evidential foundations of probabilistic reasoning*. New York, NY: John Wiley & Sons.

Tecuci, G., and Kodratoff, Y. (1990) Apprenticeship learning in imperfect domain theories. In Kodratoff, Y., and Michalski, R.S., eds., *Machine Learning: An Artificial Intelligence Approach*, Vol. III, 514–551. San Mateo, CA: Morgan Kaufmann.

Thrun, S.B., et al. (1991) The MONK's problems: a performance comparison of different learning algorithms. *Computer Science Reports*, CMU-CS-91-197. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Thrun, S., and Mitchell, T.M. (1995) Learning one more thing. *IJCAI-95*, 1217–1223.

Torgo, L. (1993) Controlled redundancy in incremental rule learning. *Proceedings of the European Conference on Machine Learning*, 185–195.

Utgoff, P.E. (1988) ID5: an incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning*, 107–120.

Utgoff, P.E. (1989) Improved training via incremental learning. *Proceedings of the Sixth International Workshop on Machine Learning*, 362–365.

Watanabe, L., and Elio, R. (1987) Guiding constructive induction for incremental learning from examples. *IJCAI-87*, 293–296.

Weiss, S.M., and Kulikowski, C.A. (1992) *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning and expert systems*. San Mateo, CA: Morgan Kaufmann.

Widmer, G. (1989) An incremental version of Bergadano & Giordana's Integrated Learning Strategy. *Proceedings of the Fourth European Working Session on Learning*, 227–238.

Widmer, G. (1996a) On-line metalearning in changing contexts: MetaL(B) and MetaL(IB). *Proceedings of the Third International Workshop on Multistrategy Learning*, 217–228.

Widmer, G. (1996b) Personal communication.

Widmer, G., and Kubat, M. (1996) Learning in the presence of concept drift and hidden contexts. *Machine Learning* 23:69–101.

Winston, P.H. (1975) Learning structural descriptions from examples. In Winston, P.H., ed., *The Psychology of Computer Vision*. New York, NY: McGraw Hill.

Wnek, J. (1996) Personal communication.

Wnek, J.; Kaufman, K.; Bloedorn, E.; and Michalski, R.S. (1995) Selective induction learning system AQ15c: the method and user's guide. *Reports of the Machine Learning and Inference Laboratory*, MLI 95–4. Machine Learning and Inference Laboratory, Department of Computer Science, George Mason University, Fairfax, VA.

Appendicies

Appendix A:
Learning Parameters
for the Computer Intrusion Detection Problem

AQ parameters

run	mode	ambig	trim	wts	maxstar	echo	criteria	verbose	test
1	ic	pos	gen	cpx	10	pv	default	1	m

AQ-PM parameters

run	learning	mode	ambig	trim	wts	maxstar	echo
1	partial	ic	pos	gen	cpx	10	pv

criteria	verbose	useinhypos	intersect	update	forget	test
default	1	yes	borders	accum	off	m

GEM parameters

run	mode	ambig	trim	wts	maxstar	echo	criteria	increment	full
1	ic	pos	gen	cpx	10	pv	default	10	y

AQ11 parameters

run	mode	ambig	trim	wts	maxstar	echo	criteria	increment	full
1	ic	pos	gen	cpx	10	pv	default	10	n

Appendix B:
Learning Parameters
for the Email Learning Agent (ELA) Problem

AQ parameters

```
run maxstar mode  trim  echo  ambig  wts test
1   10      ic    gen   pv    empty  cpx m
```

AQ-PM parameters

```
run   learning  mode   ambig   trim   wts   maxstar  echo
1     partial   ic     pos     gen    cpx   10       pv

criteria  verbose  useinhypos intersect  update  forget  test
default   1        no         borders    accum   off     m
```

Appendix C:
Learning Parameters
for the Blasting Cap Detection Problem

AQ parameters

```
run maxstar mode  trim  echo  ambig  wts test
1   10      ic    gen   pv    empty  cpx m
```

AQ-PM parameters

```
run   learning  mode   ambig   trim   wts   maxstar  echo
1     partial   ic     empty   gen    cpx   10       pv

criteria  verbose  useinhypos intersect  update  forget  test
default   1        yes        borders    accum   off     m
```

GEM parameters

```
run maxstar mode  trim  echo  ambig  wts increment  full
1   10      ic    gen   pv    empty  cpx 10         y
```

AQ11 parameters

```
run maxstar mode  trim  echo  ambig  wts  increment  full
1   10      ic    gen   pv    empty  cpx  10         n
```

Appendix D:
Learning Parameters
for the STAGGER Concepts

AQ parameters

```
run   mode   ambig   trim   wts   maxstar  echo   criteria  verbose  test
1     ic     pos     gen    cpx   10       pv     default   1        m
```

AQ-PM parameters

```
run   learning  mode   ambig   trim   wts   maxstar  echo
1     partial   ic     recent  gen    cpx   10       pv

criteria  verbose  useinhypos intersect  update  forget  test
default   1        no         borders    accum   t50     m
```

Curriculum Vitae

Marcus A. Maloof was born on 3 February 1996 in Ducktown, Tennessee. After graduating from Fannin County Comprehensive High School, located in Blue Ridge, Georgia, in 1984, Mark attended the University of Georgia in Athens. He completed a Bachelor of Science degree in Computer Science in 1989 and a Master of Science degree in Artificial Intelligence in 1992. His Master's thesis, under the direction of Krys Kochut, dealt with incorporating temporal reasoning mechanisms into a production system. After arriving at George Mason University in Fairfax, Virginia, in the Fall of 1992, Mark spent three semesters teaching introductory computer science courses. In the Fall of 1993, he became a research assistant in the Machine Learning and Inference Laboratory and began working with Ryszard Michalski. In the Fall of 1994, Mark taught the senior-level introductory Artificial Intelligence course at Georgetown University. While at the Machine Learning and Inference Laboratory, Mark investigated how machine learning methods apply to problems in computer vision, in particular, to the problem of shape recognition. He was also interested in the development and study of systems that learn over time. This dissertation was defended publicly on 25 November 1996.