The XCS Classifier System in a Financial Market

Erin K. Boland
Chemical Engineering Dept.
121 Fenske Laboratory
Pennsylvania State University
University Park, PA 16802

Karl R. Klingebiel
Aerospace Engineering Dept.
234 Hammond Building
Pennsylvania State University
University Park, PA 16802

Theodore R. Stodgell
Aerospace Engineering Dept.
233 Hammond Building
Pennsylvania State University
University Park, PA 16802

Abstract

This paper investigates an application of the XCS learning classifier system to a stock market environment with the goal of executing stock trades for profit. The effects of varying XCS system parameters are first investigated in a set of trade studies. Improved settings were found for a maze environment and XCS learning phases were characterized. System performance was then evaluated on stock market environments. The results indicate that though XCS may be able to trade profitably in a financial market, several challenges slow the evolution of high-performing problem solutions. There are enough independent factors which affect performance, including environment sense string formulation, reward schema and algorithm parameters, which may inhibit the learning ability of XCS in a stochastic environment.

1 INTRODUCTION

The stock market is a tempting investment environment, giving a savvy investor the potential to make a significant amount of profit. Because of the large potential for profit, a noteworthy effort has been placed into modeling and predicting the trends of the stock market in addition to portfolio risk management and masking money flow in an attempt to hide trading moves. Though many of the advanced applications are proprietary, there have been many efforts made using genetic algorithms and classifier systems in an attempt to model financial markets.

One strategy (Lin et al. 2004) presents a genetic algorithm that overcomes the problem of variable market trading parameters by applying a simple two-step divide-and-conquer approach. The algorithm first chooses a sub domain and then finds a near optimal value within it. The sub domains are then analyzed to find the optimal parameters over the entire trading domain. In another study, Kaboudan (Kaboudan 2004) predicts stock prices using genetic programming (GP) to develop a profitable trading strategy and then proceeds to demonstrate whether processes are GP-predictable. A model to produce one-day-ahead forecasts is evolved and proven over a fifty-day investment period.

In addition to being treated as a system which can be described mathematically, the stock market can be evaluated as a population of trading agents. Based on Fama's definition (Fama 1965), an efficient marketplace must fulfill two key criteria: the individual agents must be classified as rational and all information must be freely available. A rational agent is an individual who seeks to maximize their economic value or profit, each agent will actively attempt to predict future market trends based on the available information and behave accordingly. In this scenario, the actual economic value of a security reflects all information concerning its economic environment at that point in time. Given the speed of information transmission over the internet and the accessibility online brokers, treating the stock market as an efficient marketplace is not an unrealistic assumption.

Learning classifier systems have been successfully applied as trading agents in a simulated marketplace environment (Schulenburg and Ross 2000). In one experiment, three types of agents were devised: a non-intelligent agent that practiced a buy-and-hold strategy, a bank agent that placed all of its wealth in a savings account with 8% interest and an intelligent trader agent with access to a source of information concerning the state of the financial market. None of the agents could change their identities, though they could change their goal from maximizing their own wealth to mimicking the behavior of another successful agent. In a study consisting of three trader agents, a bank agent and a buy-and-hold agent, the three intelligent agents always outperformed the others.

Zhou and Purvis (Zhou and Purvis 2004) have developed a system called market-based rule learning (MBRL). The technique uses modified classifier systems to refine trading rules that have been extracted by neural networks, and to possibly discover new, higher performing rules. The classifier-system-based model used in MBRL attempts to eliminate some weaknesses of classic classifier systems such as difficulty in initial classifier generation and system parameter settings. In addition,

MBRL seeks to generate rule sets which are more easily interpreted by the user than those generated by classical systems. Though the study is not intended to be financial, its application in the financial marketplace is a straightforward extension.

If the stock market is treated as an efficient marketplace, a learning classifier system could be used to evolve an optimal rule set to govern the behavior of an autonomous agent operating in a stock market environment. The treatment of the stock market as an efficient marketplace is an important decision. Close-of-business prices and price derivatives will be used in the learning classifier environment implementation; these data must be expected to reflect the behavior of intelligent fully informed agents. If these economic assumptions are not valid, a learning classifier system could not be expected to make the best choices because the agent behavior it studies and attempts to mimic would not be based on the most intelligent decisions possible.

The current study attempts to apply an accuracy-based classifier system (XCS) with a unique stock-market environment formulation to develop a profitable set of trading rules. To our knowledge there is no prior documentation of this approach. The following discussion presents an improvement on XCS operational parameters in a maze environment and the results of a stock market implementation.

1.1 THE XCS CLASSIFIER SYSTEM

The XCS classifier system was first proposed by Stewart Wilson (Wilson 1995). XCS is based on the Michigan-type classifier systems originally implemented by Holland (Holland and Reitman 1978), but includes a number of improvements. Geyer-Schulz's complete description of Holland classifier systems will be summarized before the description of Wilson's improvements (Geyer-Schulz 1995).

The goal of a classifier system is to use information about its environment to generate actions on that environment in order to receive the highest payoff. Environments may fall into one of two classifications: single-step or multi-step. Single-step environments provide external reward to the system on every time-step and the environmental state at each time-step is independent of previous states. An example of such an environment would be a Boolean multiplexer problem which represents a set of logic gates. In this environment a single set of classifiers matches a single binary output for a given binary string input.

Multi-step environments do not necessarily produce an external reward for the system on every time-step and the current environmental state may be dependent on previous states and actions. The stock market is an example of a multi-step environment. A classifier system implementation for stock trading would use market data such as current stock prices, to make trading in order to maximize profits. In this case, the environment evolves throughout the trading period. Each member of the rule set which governs the decisions to act on the environment is called a classifier. A classifier takes the form of an if-then statement as often seen in computer programming. Every classifier has a condition which represents a range of environment states. If the environment meets the classifier's condition the classifier will propose an action on the environment. In addition to the set of classifiers, the Holland classifier system contains three other main components: the production system, the apportionment of credit system, and the rule discovery system.

The production system provides communication between the classifiers and the environment interface (sensors for reading the environmental state, and effectors for performing actions on the environment). Classic classifier systems implement the production system with a "message list" where sensors post messages containing the environmental state and classifiers post messages proposing actions. The message list can be read by classifiers and effectors. Based on the current messages, the classifiers and effectors determine if they should act.

The apportionment of credit system determines which classifiers are most responsible for actions that have produced positive payoff. A strength parameter is associated with every classifier and can be modified by the apportionment of credit system. One use of this value is in determining which classifiers are allowed to become active. However, the message list is restricted in the number of messages which may exist at a given time, so only a subset of classifiers with satisfied conditions are allowed to become active and post a message to the list. Classifiers compete for the chance to be active in an auction type system where their strength can be thought of as wealth. The classifiers which are able to make the highest "bids" in an auction have the best chance of becoming active. When a payoff from the environment is received, it is distributed among the winning classifiers as modifications to their strength values. This method of assigning credit to the classifiers is known as the bucket brigade algorithm.

Development of new classifiers is handled primarily by the rule discovery system. Following an initial random generation of classifiers, a genetic algorithm (GA) is run periodically using the set of classifiers as its population and the strength value of each classifier as the fitness value. The goal of the GA is to improve the quality of the population of classifiers over a number of generations through the evolutionary concepts of mating, mutation and selection. This method makes the assumption that a classifier's strength is a good representation of its fitness.

In addition to the evolution of classifiers through the GA, there are three operators which can also create new classifiers. These are known as cover operators and the triggered-chaining operator. In the case that there are no classifiers which have conditions satisfied by the current sensor readings, the cover detector operator will create a new classifier having a condition that is satisfied. Similarly, the cover effector operator creates new classifiers when none of the effectors are being activated

by the current set of classifiers and sensor readings. The triggered-chaining operator is implemented to produce pairs of classifiers in which the condition of one classifier is satisfied when the other becomes active. The purpose of these pairs is to enable short term memory in the system.

Wilson made the observation that the use of classifier strength as fitness led to the system favoring high payoff classifiers. The GA will often evolve the population to include large numbers of classifiers which exist in high-payoff niches. Additionally, classifiers which are overly general but have the same average payoff as accurate classifiers are favored. This tendency often causes the deletion of classifiers governing behavior in low-payoff niches. This behavior is a problem since low-payoff classifiers may be the best choice in particular environmental regions and retaining these classifiers may lead to better overall performance. In order to mitigate this problem Wilson developed the XCS classifier system, incorporating two major changes.

In XCS, the most significant change to classic classifier systems is the replacement of the strength parameter by a set of three new attributes: prediction, prediction error and fitness. The prediction parameter p represents an estimate of future payoff based on the past rewards a classifier has earned. The prediction error ε represents the error between predicted payoff and actual payoff. Finally, fitness F is defined as the classifier's accuracy (an inverse function of the prediction error). This arrangement allows the system to discriminate between parameters for determining which actions to take and which classifiers are most fit. The key point is that in XCS, the GA evaluates and evolves classifiers based on their accuracy in predicting payoff, rather than the magnitude of their payoff.

The second major difference between classic classifier systems and XCS lies in the apportionment of credit system. Instead of the bucket brigade algorithm acting upon a single attribute (strength), classifier attributes in XCS are adjusted by a suite of algorithms including Q-learning methods (Watkins 1992) and the *moyenne adaptif modifée* (MAM) technique (Wilson 1995). Rather than reinforcing strength or payoff, the algorithms adjust P, ε and F. Classifier attributes "learn" or are adjusted based on their own experience as well as the collective experiences of like classifiers.

A more mathematically rigorous explanation of XCS is beyond the scope of this discussion but may be referenced in Wilson's 1995 paper. However, a high-level overview of some key attributes and parameters in XCS follows with the understanding that this paper cannot treat XCS in utmost detail.

1.2 XCS CLASSIFIER ATTRIBUTES

As described previously, each classifier has three attributes: p, ε and F used in the XCS reinforcement learning methods and the GA. Additionally, each classifier carries a fourth attribute n to represent the concept of *numerosity*, used to reduce computation time by minimizing population size and facilitating matching. XCS does not permit duplicate classifiers to coexist in a population. If the GA should happen to evolve a child classifier with the same sense-action pair (considering wildcards) as a parent, XCS deletes the less general of the pair and increments the second classifier's numerosity by one. All classifiers begin life with a numerosity of one. System operations acting upon a macroclassifier, a classifier where n is greater than 1 treat the macroclassifier as n individual instances.

1.3 XCS SYSTEM PARAMETERS

To enhance understanding of the XCS algorithm and ease interpretation of performance results a brief discussion of user-controlled parameters is provided below.

1.3.1 Learning and Adjustment Parameters

XCS permits the user to fine tune the learning process via the set of related parameters, α, β, γ, ε_0, and ν. The rate at which at which P, ε and F are adjusted for experienced classifiers is set by α. The discount factor γ controls the proportion of knowledge gained from its own experience versus the experience of other classifiers. For young classifiers which have not yet participated in $1/\beta$ action sets, a simplified accelerated learning technique is used to rapidly adjust P, ε and F to meaningful values. ε_0 sets the error discrimination threshold between classifiers; classifiers which differ in error less than ε_0 are considered to have equal error. The exponent ν is applied in an internal function which scales errors nonlinearly. Table 1 displays generally appropriate ranges for selected parameters.

Table 1: XCS Learning and Adjustment Parameters

Parameter	Absolute Range	Suggested Optimal Range
α	$0 < \alpha < 1$	0.1 – 0.3
β	$0 < \beta < 1$	0.1 – 0.2
γ	$0 < \gamma < 1$	0.1 – 0.9
ε_0	Reals > 0	10^{-2} – 10^{1}
ν	Integers > 1	5

1.3.2 Genetic Algorithm Parameters

The GA used in the XCS implementation chosen for this study is a traditional binary-coded genetic algorithm. It generates children via mutation and uniform crossover operators, choosing fit individuals though tournament selection. The mutation probability P_M, crossover probability P_X, and the probability for generating wildcard-bits $P_{dontcare}$ may be set by the user. A wildcard-bit, represented by a hash-mark (#) will match with either

a 1 or a 0 in the environment sense string resulting in a more general rule. The value for $P_{dontcare}$ is usually set as a function of the sense-string size.

1.3.3 Population Parameters

The classifier population is permitted to grow or shrink as XCS runs however a hard limit to the population size exists. An optimal value for the maximum population size is highly dependant on the complexity of the environment and the number of possible sense-action combinations. Subsets of the entire population which match the current sensory input string are known as match sets. A user-definable constant, θ_{GA}, controls how many time-steps pass between applications of the genetic algorithm to the current match set. In a multi-step environment, an appropriate value for θ_{GA} is largely defined by the number of steps possible in the environment.

As previously described, XCS uses the concept of numerosity to eliminate duplicate classifiers. If, however, two classifiers are identical with the exception of their wildcard bits (#), the more general classifier may subsume or absorb the less general one. In subsumption the more general classifier's numerosity is incremented while the more specific classifier is deleted. Three user-defined parameters permit adjustment of subsumption: $\theta_{subsumption}$, *GA-subsumption*, and *Action-Set-Subsumption*. $\theta_{subsumption}$ sets the minimum age at which a classifier becomes eligible for subsumption. *GA-subsumption* and *action-set-subsumption* are simple Booleans. *GA-subsumption* determines whether individual classifiers are permitted to subsume other individual classifiers and *action-set-subsumption* determines whether entire action sets of classifiers are permitted to subsume other action-sets.

1.3.4 Classifier Initialization Values

At the start of operation, or when a new random classifier is generated via covering, the system must supply initial values for F, ε and p. These initial values are set as three user-definable constants: F_I, ε_I and p_I whose ranges are arbitrary. The initial, completely random population of classifiers is assumed to be suboptimal.

2 IMPLEMENTATION AND TESTING METHODOLOGY

For this study, Butz's JavaXCS 1.0 implementation was chosen. The algorithm's performance was analyzed in a maze environment using trade studies over a range of parameter values. A trade study involves varying one parameter across a wide range while holding all other parameters constant, enabling a view of the parameter's influence on the algorithm's behavior. In the case of a traditional genetic algorithm, the number of function evaluations is most commonly used as a performance metric. This is inappropriate for XCS. Instead, prediction accuracy and an environment defined performance metric are preferred. In the case of the maze environment, the performance is evaluated as the number of steps required to reach the food, while in the stock market, performance is evaluated in terms of the agent's final wealth.

During each trial XCS performs in two different modes: exploration and exploitation. During the exploration mode, XCS makes random moves in an attempt to learn as much as possible about its environment. During the exploitation mode, XCS uses what it has learned in an attempt to maximize environmental reward. In all analyses, data are reported as results of the exploitation mode runs.

For the trade studies XCS was allowed to make 4950 runs through its environment, learning as it proceeded. By the final set of trials, the classifier system was expected to be fit. The data are reported as the average of the final fifty environment trials. To minimize variation in the data trends, each trade study is averaged over ten different random starting populations. In cases where the out-of-the-box parameters from Butz's original source code were found to be sub-optimal, second and third generation trade-studies elucidated trends in algorithm's performance. For the sake of brevity, only three of the most important trade studies and a learning rate analysis have been selected for discussion. For the application of XCS to the stock market environment, Table 2 shows the parameters which were changed from Butz's original source code. A discussion of the motivation for these changes is provided in section 3.

Table 2: Parameter Comparison: Butz vs. Stock Market

Parameter	Butz	Stock Market
γ	0.95	0.1
δ	0.1	0.95
ε_0	10	0.002
θ_{del}	20	30
P_M	0.04	0.2
$P_{dontcare}$	0.5	0.05

2.1 MAZE ENVIRONMENTS

The maze environment is a typical multi-step environment used to test performance in a learning classifier system. Though the maze is a relatively small environment, it is complicated, with few large blocks of open space. Figure 1 shows a pictorial representation of a typical maze. Table 3 is provided as a key. The classifier system models an intelligent agent, or animat moving through the maze searching for food. In addition to the spatial definition of the environment, the animat is also given functionality to perform actions within its environment. The animat can move about in empty space and perceive its immediate surroundings.

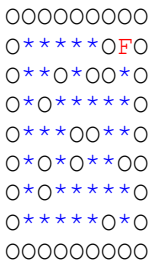

Figure 1: Sample Maze Environment

Table 3: Environment Symbol Key

SYMBOL	DESCRIPTION
*	Open Space
O	Obstacle
F	Food

A three-bit maze environment (24 bit sense string) with 8 movement actions has $8x2^{24}$ or $1.342x10^{8}$ unique sense-action pairs. Disregarding don't-care bits, XCS does not permit multiple classifiers to have matching sense-action pairs, so a theoretical extreme upper limit of population size exists. In practical applications, though, XCS would never be implemented with populations even remotely this large.

2.2 ONE-BIT STOCK MARKET ENVIRNONMENT

The stock environment sense string consists of binary digits representing the information that an agent may use to make a decision. The implementation has two input formats, a real number describing the current state of a particular stock and a binary string describing the state of the environment with respect to past information. The real input allows the system's detectors and effectors to make accurate movements within the environment. The binary string allows classifiers to be general with respect to the stock identity, allowing a rule set developed on a stock with a unique range of values to be used on a stock with an entirely different range. Since the fitness in XCS is accuracy based, the environment could potentially consist of real numbers represented in binary encoding; however this would result in a very large environment and slow down the learning rate immensely. This study does not include an analysis of this representation.

In the ten-bit long environment, the first eight bits represent market information and the last two bits will be added by the market environment implementation. The added bits will represent whether or not all of the agent's money is located in the bank and whether selling owned stock would lead to a profit after commission is paid. Table 4 describes the environment variables that XCS has access to and Table 5 describes the environmental representation of pertinent information. Subscripts t, t-1, and t-2 indicate present and past time. Currently the environment contains no hysteritic information concerning whether or not a sale would be more profitable today then it was yesterday.

Table 4: Stock Market Environment Variables

Variable	Definition
P	Stock Price
D	Price Derivative
P_{10}	Volume-Weighted 10-Day Moving Average
D_{10}	Moving Average Derivative

Table 5: Stock Market Binary Environment

Bit	Representation
1	$P_t \geq P_{t-1}$
2	$P_{t-1} \geq P_{t-2}$
3	$D_t \geq D_{t-1}$
4	$D_{t-1} \geq D_{t-2}$
5	$P_{10t} \geq P_{10t-1}$
6	$P_{10t-1} \geq P_{10t-2}$
7	$D_{10t} \geq D_{10t-1}$
8	$D_{10t-1} \geq D_{10t-2}$
9	All Money in the Bank
10	Is a Sale Now Profitable?

The stock market environment has a ten-bit sense string with three possible actions: buy, sell or hold. This translates to a maximum of $3x2^{10}$ or 3072 unique sense-action pairs, a much smaller theoretical figure then the three-bit maze environment. This is a much smaller set then would exist if XCS were given individual binary-encoded variable values in 16- or 32-bit representation. Given this fact, XCS should not require as large of a population as a multi-step maze environment. This representation scheme also has several advantages; it is very straightforward to add or subtract information from the environment. It also imparts a flexibility to the rule set that XCS generates: a set that performs well for a high-cap stock should, in theory, perform well for a low-cap stock with similar overall trends.

In the stock market environment, XCS can take three possible actions: buy, sell or hold. Currently the classifiers are rewarded for a sale that makes money. If a real encoded environment is used, a scheme that returns a change in wealth may be the most appropriate reward. Several combinations of environmental variables were

tried, with combinations including information on price and derivatives. The environment formalism presented in Table 5 is the first environment that worked well: it consists of two day derivative information and ten-day moving average price information. While this particular environment did provide profitable results with a simple rewards scheme, it may not be optimal. For the sake of this study, unless otherwise noted – this is the environment used to analyze the XCS stock agent performance.

3 TRADE STUDY RESULTS

A key performance metric in XCS is the rate at which the classifier population maps the payoff from the environment. Figure 2 shows that XCS improves its maze performance continuously right up to the specified stopping criterion. Due to computation time the trade studies were limited to 5000 runs; the impact of XCS parameters upon performance is still clear. If XCS had been allowed many thousand more maze runs with which to learn, the trends would eventually asymptote to an almost constant number of steps to reach the food.

The two data sets in Figure 2 show maze performance at near optimal settings and average out-of-the-box parameters. Each point represents the average performance of 50 maze runs in exploitation mode.

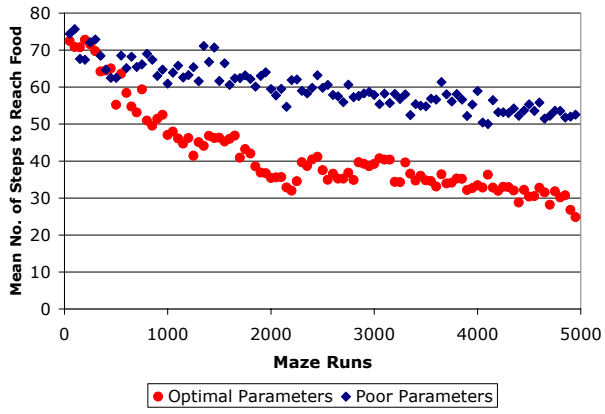

Figure 2: Learning Rate – Steps to Food vs. Maze Runs

Figure 3 shows the learning rate in terms of the average prediction error, ε, and compares it to the steps required to reach the food. The average prediction error settles quite rapidly, long before the overall system begins to exhibit high performance. Learning in the life of an XCS classifier population can be characterized into two phases as seen in Figure 3. Before learning begins, there is an initial period of equilibration or rapid adjustment where the majority of classifiers are grossly inappropriate. In this initial period, roughly between 0 and 100 maze runs, the classifiers perform very poorly and are unaware of their poor performance.

The next phase in the life of a classifier population appears once most classifiers have gained enough experience to predict error well. Though the classifiers have not yet learned high performance actions, most classifiers have a baseline of experience. With only moderate payoff, classifiers can predict their own performance accurately. This stage appears in the maze environment roughly between 100 and 2000 runs.

Finally, the classifiers slowly evolve towards higher performance actions while the average prediction error remains stable. The classifiers learn to perform better while continuing to predict their performance well.

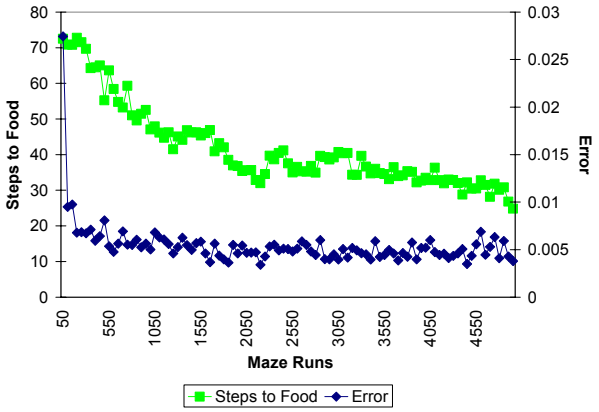

Figure 3: Learning Rate – Average Prediction Error and Steps to Food vs. Maze Runs

The fitness adjustment parameters ε and γ were found to be critical in achieving the best performance with XCS. Figure 4 shows the effect of the error discrimination value ε_0. All classifiers with an error prediction value greater than ε_0 are considered to be equally unfit and their experiences are excluded from the fitness adjustment algorithm. Classifiers with small prediction error are preferred and their experiences are used in adjusting the fitness of other classifiers.

An overview of the reinforcement learning method used in fitness adjustment is shown in Equations 1-4 below. A more detailed explanation is available in Lanzi's notes (2002). Every *j*-th classifier's fitness is adjusted via both its own prediction error and collective data from other classifiers in the same action set.

$$\left\{\varepsilon_j > \varepsilon_0\right\} : \kappa_j = \alpha\left(\frac{\varepsilon_j}{\varepsilon_0}\right)^{\nu} \qquad [1]$$

$$\left\{\varepsilon_j \le \varepsilon_0\right\} : \kappa_j = 1 \qquad [2]$$

$$\kappa_j' = \kappa_j \Big/ \sum \kappa_j \qquad [3]$$

$$F_j \leftarrow F_j + \alpha(\kappa_j' - F_j) \qquad [4]$$

Setting ε_0 too high excludes classifiers from collective reinforcement learning causing a classifier's fitness to be adjusted only through its own individual experiences. Setting ε_0 too low includes poorly predicting classifiers in the reinforcement learning method. This causes all of the classifiers' fitness to be perturbed by including low-worth information from poor-predicting classifiers. The sweet spot in the curve in Figure 4 coincides with ε_0 approximately one order of magnitude greater than the average steady-state error prediction value. Any classifier with error greater than one order of magnitude above the population's average error is excluded from the fitness reinforcement learning process. In the case of the stock market implementation, ε_0 was chosen using this principle.

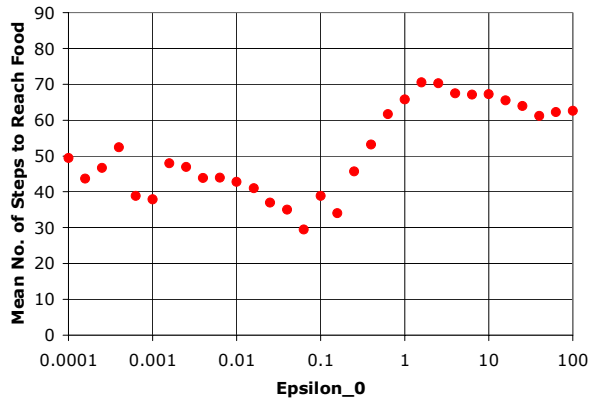

Figure 4: Steps to Food vs. ε_0

Figure 5 shows the effect of the adjustment discount factor, γ, which plays a role in the adjustment of prediction, and prediction error. A classifier's values of P and ε are adjusted with data from both that classifier's own experiences and the experiences of other classifiers in the same set. As with ε_0, γ can be thought of as a learning adjustment parameter which balances the weight of a classifier's *own* experiences against the collective experience of other classifiers in the same set. The highest performing value for γ found in the trade studies for the three-bit maze environment was 0.1. Butz suggests 0.71 (Butz and Wilson 2001), though it is mentioned that the optimal value may vary with the environment. To minimize the number of independent variables that would affect performance, parameters were not re-optimized for the stock market environment. A value of 0.1 was chosen for γ because it produced the optimal results in the maze environment.

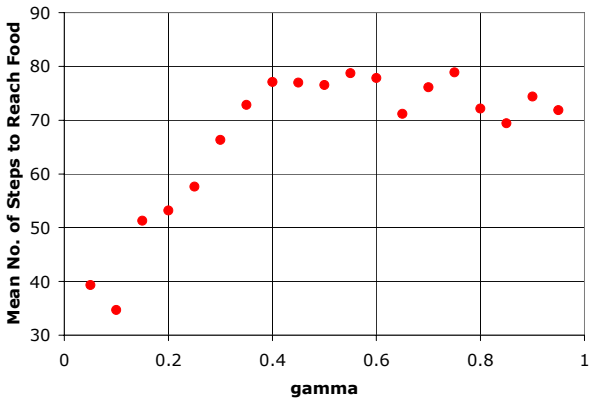

Figure 5: Steps to Reach Food vs. γ

The average prediction error increases with increasing γ, as shown in Figure 6. Learning can be more difficult when classifiers are forced to learn mostly by their own experiences without sharing feedback among like classifiers.

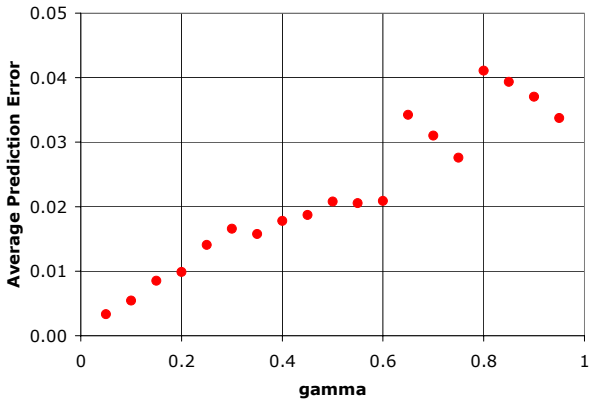

Figure 6: Average Prediction Error vs. γ

The impact of the exponent ν (see Equation 1) used in adjusting classifier fitness is shown in Figure 7. This exponent scales the difference in prediction error when comparing fit classifiers. A value of 5 was found to be appropriate.

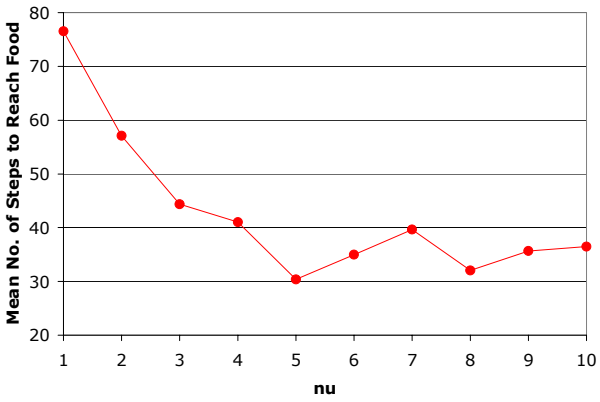

Figure 7: Steps to Food vs. ν

4 RESULTS AND DISCUSSION

The stock market environment implementation was run on four data sets: two historical 5-year-long stock data sets (for 2000 – 2004 trading years) and two test functions designed to represent an idealized set of stock data. The trading agent began each trial with a $10,000 trading account. Interest was not earned on money left in the bank in order to simulate a real trading account. A commission of $10.00 was charged for each transaction. In all cases, performance is gauged as a function of the trader's final wealth after the 5-year period.

Harrah's Entertainment (HET) was chosen because it was an upward trending stock, while Ford Motor Company (F) exhibited a steady downward trend as shown in Figure 8. Despite their overall trends, both Ford and Harrah's exhibit local price fluctuations on a small time scale. An astute trader could take advantage of local trends with well timed market maneuvers. Both stock data sets run the entire 5 year period without splitting. A company may split its stock shares when the price grows high: doubling the number of shares and splitting the share price in half. The current code has not yet been expanded to handle stock splits; splits are infrequent events and their presence may inhibit classifier learning.

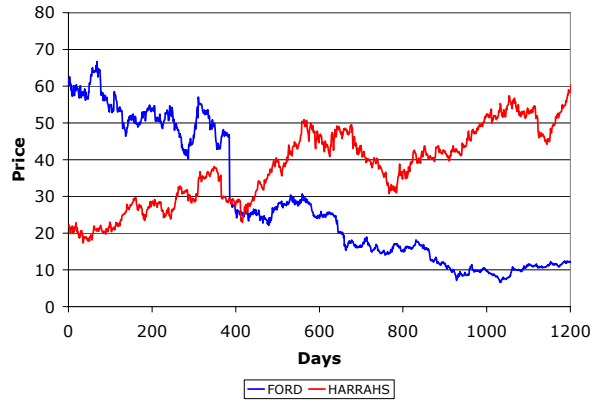

Figure 8: Stock Price Profiles

The two test functions simulated a stock with a bound price that fluctuated as a sine wave with a steady period. Illustrative portions of the test function price profiles are shown in Figure 9.

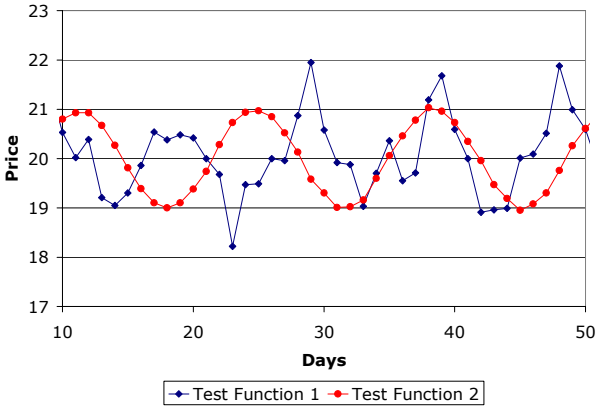

Figure 9: Test Function Price Profiles

The smoother function (Test Function 2, TF2) was infused with noise using a cubic distribution to vary the price by ±10¢. The other test case (Test Function 1, TF1) has a uniformly random noise spike of ± $1.00. Analysis of a uniformly smooth sine wave is not included because XCS was able to exploit the regularity and make the maximal profit in almost every run. Since real stocks do not perform with the uniformly predicable behavior of a sine wave, a smooth sine curve would not be representative of a real world data set. Test functions were generated and compared to historical data because it is very rare to find a stock for which price falls within a bound range for long periods of time. Such simplistic test functions serve as a good metric for XCS performance. The difference in noise between the two cases should shed light on how the system responds to varying degrees of noise in the environment data.

To map the learning rate of XCS on stock market environments, two trials using the HET and TF1 data sets were performed for 500,000 runs. Since the results were similar, only those for TF1 are presented. It is apparent from Figure 10 that there is no improvement during 500,000 trial runs, yet in terms of pure market performance, XCS consistently does well. It was infeasible to extend this study further due to a lack of computational resources.

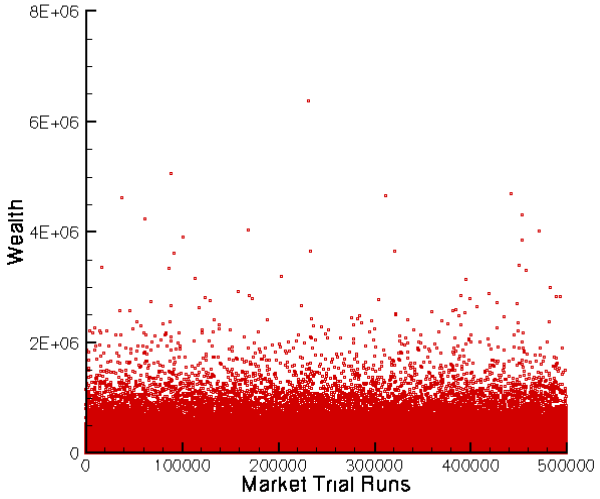

Figure 10: Final Wealth vs. Market Trial Runs

The standard deviation of the final wealth was analyzed in sequential samples of 500 trials. Figure 11 shows the results as a function of each sequentially numbered sample. With the exception of several outlying samples, the standard deviation is essentially constant. If XCS were exhibiting a learning trend, the standard deviation would be expected to decrease with subsequent trial runs. For the sake of this paper, XCS is considered to exhibit no learning trends in the stock market environment.

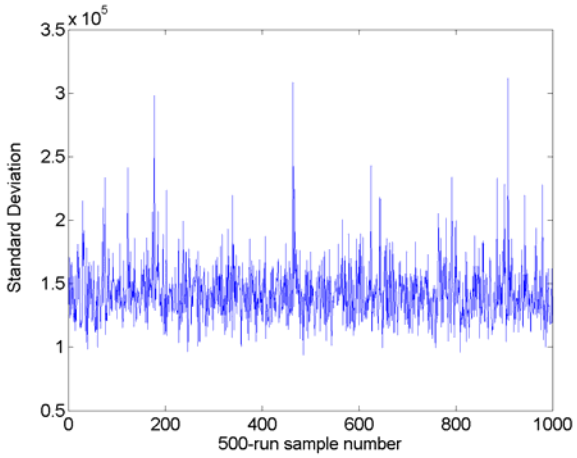

Figure 11: Standard Deviation vs. Market Trial Runs

All following studies on the stock market environment were performed with 500 trial runs through a single-stock environment. This procedure is justified because XCS shows no performance improvement with an increased trial size. Since the length of the data-set should not affect the learning over a long period of time, only five years of historical data were considered.

One of the first gauges of performance is a comparison of the four test cases. A comparison of the two historical stocks shows that the XCS trading agent performs significantly better on an upward trending stock than it does when trading a downward trending stock, though the performance is much less consistent (see Table 6 and Figure 12). When trading HET, the agent performs profitably most of the time, however it does not, on average, outperform the market as represented by the buy-and-hold strategy. Dividends have not been included in these numbers. On the trial with a downward trending stock, the agent never makes a profit, though there are many cases in which it simply chooses not to trade.

Table 6: Stock and Test Function Performance

Case	Buy and Hold	Average Wealth	Standard Deviation	Deviation Relative to Wealth
HET	30719.13	25614.16	5907.62	23.06%
F	2405.25	3832.53	2694.33	70.30%
TF1	10558.30	91781.08	127291	138.69%
TF2	9819.75	21200000	17166800	80.96%

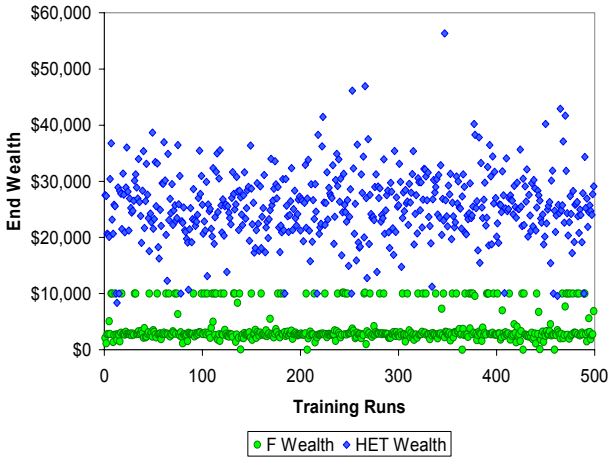

Figure 12: Historical Stock Performance

The trading agent performance on the test functions is interesting. The noisier price series, TF2, resulted in poorer performance with a much higher variance as exhibited in Figure 13. This result may explain the variance seen on historical stock price-series where prices reflect the stochastic nature of the market. It is interesting to note that the performance was three orders of magnitude better when trading TF2 than TF1, though the prices were bound in similar regions. This phenomenon could be an effect of XCS exploiting the regular pattern present in TF2.

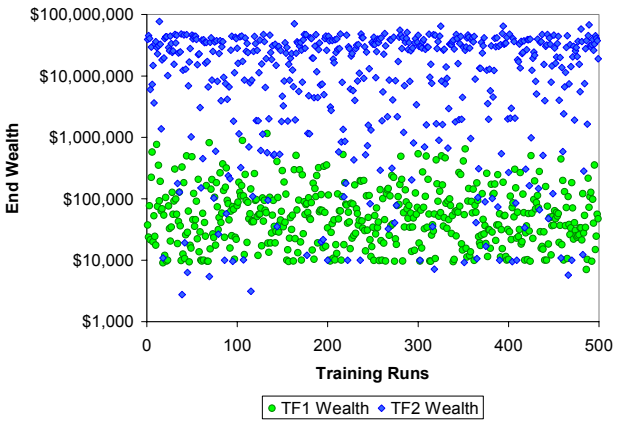

Figure 13: Test Function Performance

The trading agent's habits were analyzed for both HET and TF1 to gain insight to the agent's holding-patterns. Figure 14 shows the transactions made during the 5-year trading period. Though individual transactions are difficult to discern, a general trend is apparent: the agent usually day-trades, in most cases holding stock for no more than 4-day periods. There is one downward trending period of note, where the agent held the stock for a significant period of time: between days 348 and 425. In this case, the agent chose not to sell during the downward trending period in an attempt to minimize losses. In general, the agent typically sells a stock as soon as a profit may be made, resulting in a higher frequency of trades during a bull market and a lower frequency of trades during a bear market. This phenomenon may explain the extremely poor performance when trading F.

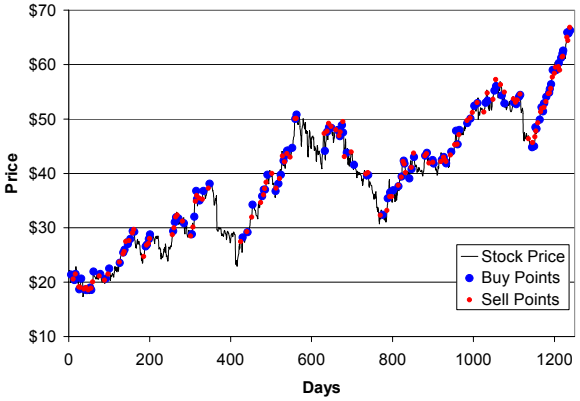

Figure 14: HET – Buy Sell Analysis

When the agent traded the TF1 stock, the holding-pattern was shorter with an average of 3 days, as shown in Figure 15. While the trader may not always find optimal points to buy and sell, there is clear evidence of buy-low-sell-high behavior. Additionally, the narrow range of price-fluctuation in this particular case allows the trading-agent to make some mistakes without suffering heavy financial losses.

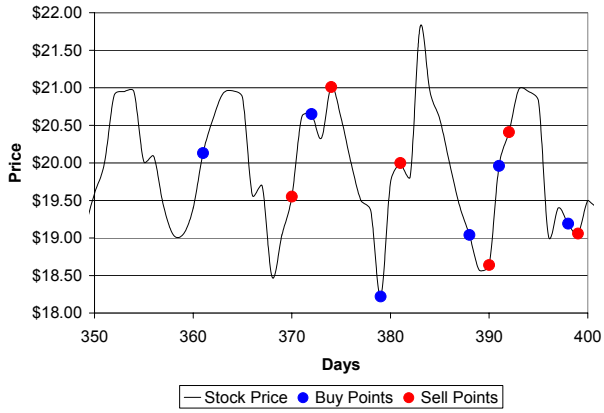

Figure 15: TF1 – Buy Sell Analysis

Gauging the performance of the trading agent is an interesting analysis point. Should the goal of XCS simply be to make a profit, or to make a profit and outperform a buy-and-hold trading strategy? If the latter is desired, the most pertinent issues regarding XCS's performance are environment and reward schema formulation. That is, how can the environment and reward schema be adapted to provide as consistent as possible performance that beats the stock market in its return on investment? A similar study, using strength-based learning classifier systems to develop optimal trading rules was preformed by Schulenburg and Ross (2000). While a one-to-one comparison of these studies is not appropriate since the approaches differed significantly. It is, however useful to discuss the differences in an effort to improve interpretation of the results presented herein. The stock studied by Schulenburg and Ross contained splits which had been handled by adjusting the price-time series. It was opted not to take this approach since it complicated the definition of moving averages. As previously mentioned, splits had not been implemented with XCS so a direct stock-to-stock performance comparison is not possible at this time.

This study presents a similarly formulated environment, where variables are represented with a one-bit binary encoding. In this formulation, the classifier system is only aware of trends and not actual values. Though the representation was the same, the environments presented by Schulenburg and Ross differ in the type of information deemed appropriate. The study also implemented a variety of reward schemes, many which seemed to possess a large degree of imposed heuristics. In contrast, the reward scheme presented here is simple: rewarding the classifier favorably only when a profitable sale is made or when a stock with increasing value is held. There are no imposed heuristics based on moving averages or price comparisons. The complexity of this type of reward

scheme seems to defeat the purpose of a learning classifier system.

The Schulenburg-Ross study does indicate that for consistently profitable performance a more complex reward method may be necessary for a 1-bit environment representation. The addition of bits representing whether or not the current price is higher than the highest historical price or lower than the lowest historical price may be a worthy extension to the environment and might cause the classifier systems to hold stock for a longer period. XCS has shown the ability to produce very profitable populations of classifiers yet it does not produce them consistently nor does it show trends of evolving more fit populations over time. Though the system generates a profit often, the results indicate that the present environment formulation, parameter settings and reward structure may not be an optimal way to develop financial trading rules. If XCS is to succeed, a reformulation of the environment or reward structure is required. Though the current reward system and environment do not allow XCS to exhibit learning behavior, they were chosen after several formulations and provided the best average performance as tested with HET.

There are several different actions that XCS can take, each with varying rewards, either negative or positive. Table 7 displays the best reward settings found. Financial maneuvers that were considered incorrect or "against the rules" e.g. trying to sell when no stock was owned or trying to purchase stock shares when no money was available were negatively rewarded moreso than maneuvers that were unprofitable. The reward scheme was discretized into integer values rather than real values representing an agent's wealth or profit because XCS is not currently given exact price (or other variable) information in this implementation.

Table 7: Stock Market Classifier Rewards

Action	Reward
Buy: No Stock is Owned	0
Buy: Stock is Owned	-1000
Sell: No Stock is Owned	-1000
Sell: Stock is Owned	100
Bank: Money in a Rising Stock	100
Bank: Money in a Falling Stock	-10
Bank: No Money in Stocks	0

Prior to implementing negative rewards, the positive rewards were investigated relative to one another. The experiments were executed on HET, since XCS showed the lowest relative variance on this data set. A description of the schema is presented in Table 8.

Table 8: Sell and Bank Reward Scheme Results

Case	Bank Reward	Sell Reward	Average Wealth	Standard Deviation
A	500	100	19932.01	5089.2
B	200	100	20205.07	4505.55
C	100	100	25472.46	3781.53
D	0	100	15664.58	6140.03

In general, the XCS stock agent performs worse with differentially weighted rewards. As shown in Figures 16 and 17, the wealth appears to be more stochastic than deterministic. It is interesting to note that as the performance improves in terms of wealth, the variance from run to run decreases. The best case shows an outlier in an early run through the market environment. This may be a pure statistical anomaly since similar outliers are found in other studies and the classifier system does not seem to develop a memory.

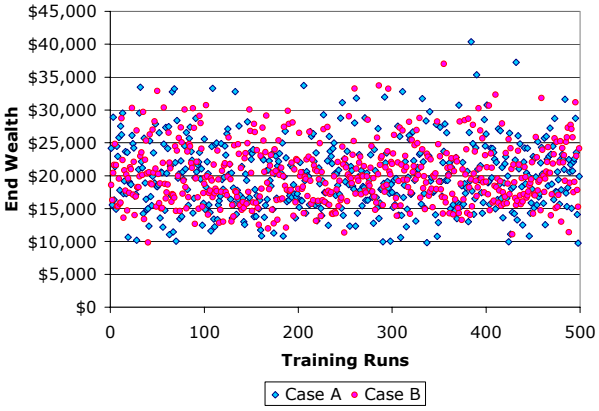

Figure 16: Wealth vs. Reward Scheme: Cases A & B

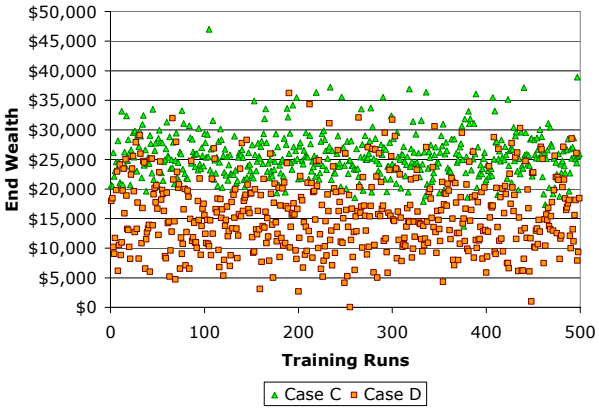

Figure 17: Wealth vs. Reward Scheme: Cases C & D

The next complication to the rewards scheme involves adding negative reinforcement when a classifier attempted actions which were defined as illegal. The addition of a moderately scaled reward of the same order of magnitude as the buy-bank rewards in Case E introduced less of a variation in the final wealth than the larger scaled reward (Case G). It did, however, result in a slightly poorer average performance. Negative rewards were then added for actions that caused the trader to lose money: selling when the price had dropped too low or holding a downward trending stock. In these cases, the performance of the classifier population tended to decrease in proportion to the negative feedback. When a slight negative signal was returned (Case F) the classifiers performance decreased from the previous best (Case G). Though the average wealth decreased slightly, the variation also decreased, making this reward schema more desirable then a case with only negative feedback for an incorrect action.

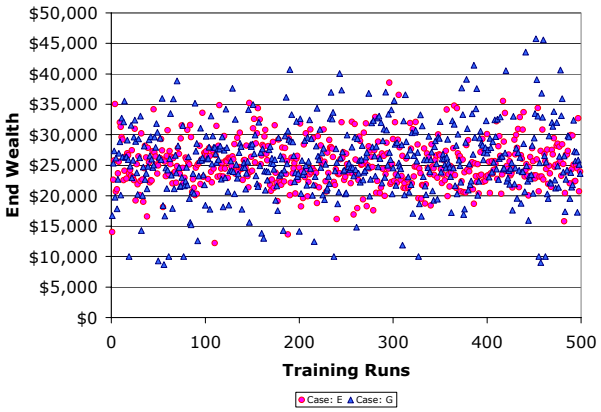

Figure 18: Reward Scheme for Incorrect Actions

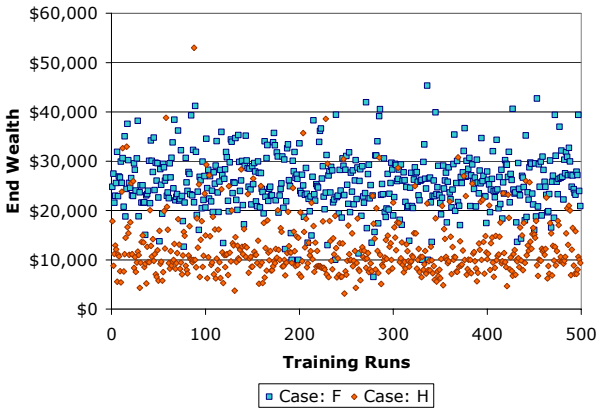

Figure 19: Reward Scheme for Unprofitable Actions

Table 9 shows that a slight amount of negative reinforcement, as in case F, can decrease variance while preserving overall profitability. Harsher penalties, however, increased variance and in case H, sharply decreased performance.

Table 9: Negative Reward Scheme Results

Case	Incorrect Buy or Sell	Poor Bank or Sell	Average Wealth	Standard Deviation
E	-100	0	25258.25	3728.27
F	-1000	-10	25654.93	5656.85
G	-1000	0	25775.86	5718.39
H	-1000	-100	12278.29	5856.62

In addition to experimenting with various reward schema, several different environment sensory configurations were tested. Table 10 shows the impact of adding extra bits of data to the sensory string. In Case 1, the agent had access to a one-day history of previous price changes. Case 2 added one extra bit, providing both one-day history and 2 day histories. Case 3 added yet another bit, encoding one, two and three day histories. The sensory string in Case 4 is equivalent to that of Case 2, with the addition of a bit to track daily change in wealth.

Over 500 training runs, XCS did not learn better performance under *any* of these cases. However, the complexity of the extra bits affected classifier performance despite the lack of learning. The case with the shortest environment string, Case 1, displayed the lowest variance yet also performed poorest. Adding additional history bits increased performance at the expense of increased variance. In general, as environments grow more complex, the rate of performance uniformity decreases. Experimental results agree with Wilson's (1998) hypothesis that learning complexity scales as a low order polynomial of the complexity of the problem, not with the complexity of the learning space.

Table 10: History Results

Case	Modified Environment	Average Wealth	Standard Deviation
1	1 Day History	24687.61	4171.33
2	1 & 2 Day History	25614.16	5907.62
3	1, 2 &3 Day History	25923.66	7529.94
4	Did Wealth Go Up?	25178.08	5394.44

The final pertinent study, inspired by the Schulenburg-Ross study examines the effect of generality on the performance of the trading agent. The original studies set $P_{dontcare}$, or the probability of including a wildcard bit in the sense string, to 0.05 or 5% of the environment sense string length. Schulenburg and Ross chose a much higher

probability – 0.5, in an effort to create a generalized rule set. The XCS results in Figure 20 and Table 11 show that the trading agent performs significantly worse when $P_{dontcare}$ is raised in an effort to generalize the trading rules. These results do not disagree with the performance in the Schulenburg-Ross study because reward heuristics were added to coax profitable trading performance.

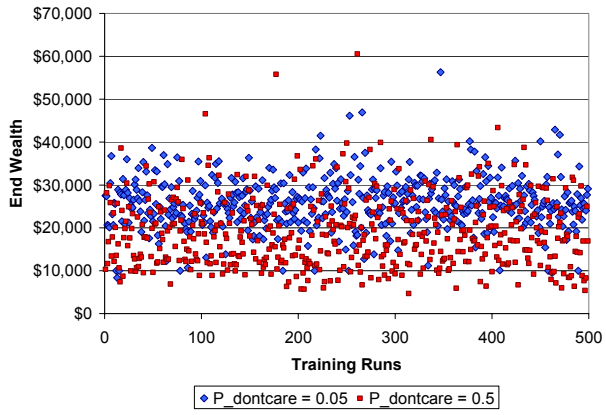

Figure 20: Reward Scheme for Unprofitable Actions

Table 11: $P_{dontcare}$ Results

Case	Modified Environment	Average Wealth	Standard Deviation
1	$P_{dontcare} = 0.05$	25614.16	5907.62
2	$P_{dontcare} = 0.5$	18926.43	8093.49

In a bull market, XCS performs slightly worse than the market average. However, one must consider that XCS still nets a profit on average and the system often comes within one standard deviation of market performance. Considering that the system is not completely optimized, results are promising. It shows that even a mediocre classifier system can exploit a purely stochastic environment.

5 CONCLUSIONS AND FUTURE WORK

The present study examines the performance of XCS in a single-option-single-stock environment as a first step towards a portfolio management system. It is clear that XCS exhibits the best potential for a profit in the case of a bound stock where it can buy low and sell high. It may not be the best application where long term investments are desired.

Before the algorithm is developed further to handle a more complex environment, extensive trade studies should be performed to find the set of XCS parameters that provide optimal performance. It may also be necessary to compare different types of environment information, for example: moving averages or different types of derivatives. XCS may have a chance to allow its accuracy based fitness metrics to perform well if the environment consists of binary encoded real values so that it has finite price and derivative information and dynamic reward schema. The tradeoff in this case would be a retarded learning rate vs. providing XCS with a non-generalized reward schema biased towards making the largest possible profit. This environment change may produce classifiers that are only valid for prices and derivatives in a specific range, resulting in a less generally usable end classifier population.

Based on these results it is clear that XCS may not be the most appropriate classifier system for handling a stock market environment. While this conclusion is preliminary and more experimentation is needed in order to assess XCS's performance, the current set of data strongly supports this hypothesis. However, XCS does occasionally produce excellent profitable outliers and each set of classifiers produced over different runs through an environment could be mined for use in a static expert system. However, if an XCS environment that successfully produced profitable trades could be implemented, there are some improvements that could be made to make it a more practical tool for stock investing.

The first step is the implementation of a multi-option-single-stock environment where XCS can choose a single type of stock to trade at any given time from a set of choices. Granting XCS the opportunity to trade among several bound stocks may provide a greater opportunity for profit maximization than may be feasible when only one oscillating stock is bought and sold. This environment would take advantage of market timing and it may be able to exploit stock increases continuously instead of waiting for an oscillating price to fall.

While the multi-option-single-stock environment may be the optimal scenario in terms of pure profit, the next step in extending the program's functionality is the implementation of a multiple-option-multiple-stock environment with the inclusion of an aggressiveness factor. In this scenario, XCS would manage a portfolio. An aggressiveness factor would describe whether or not the algorithm attempts to minimize risk or maximize profit. In this case, if risk minimization were desired, the algorithm might either choose to sit on "upper" stocks for long periods or time or diversify holdings, while a portfolio that chose to maximize profit may choose to perform similarly to a multi-option-single-stock environment, where stocks with the largest derivatives are bought and sold on a day to day basis.

The implementation of a multi-option-single-stock environment would be a fairly straightforward step in extending the program, requiring only the addition of tracking variables in the XCS environment. This implementation may even be possible by training XCS in a single stock environment and then implementing the

optimal classifier in a multi-option-single-stock environment.

Expanding the market environment to manage a portfolio is a more ambitious task than the extension of the multi-option-single-stock environment. This extension is the most complex of the two being proposed here and would require large modifications to XCS. It would be the simplest to execute in an object oriented program where a stock class could be implemented and used in a portfolio environment.

Acknowledgments

We would like to thank the Graduate Education and Research Services Group (GEARS) for their cooperation in providing joint drive space members in our group. Shared space proved indispensable in the completion of our project.

We would also like to acknowledge the Illinois Genetic Algorithms Laboratory (IlliGAL) for providing Java source code for Martin Butz's XCS and Wharton Research Data Services for providing daily stock price profiles.

References

Butz, M.V. and Wilson, S.W. (2001) An Algorithmic Description of XCS, LNAI 1996, Lanzi, P. L., Stolzmann, W., and S. W. Wilson (Eds.), *Advances in Learning Classifier Systems* (pp. 253-272)

Fama, E.F. (1966) Random Walks in Stock-Market Prices, *Financial Analyst Journal*, London.

Geyer-Schulz, A. (1995) Holland Classifier Systems, *APL Quote Quad,* Vol. 25 No. 4, pp. 43-55.

Holland, J.H. and Reitman, J.S. (1978) Cognitive systems based on adaptive algorithms. In D.A. Waterman & F. Hayes-Roth (Eds.), *Pattern Directed Inference Systems* (pp. 313-329)

Kaboudan, M.A. (2000) Genetic Programming Prediction of Stock Prices, *Computational Economics*, Vol. 16 pp. 207-236.

Lanzi, P. L. (2002) Lezione *10: Sistemi a Classificatori*, class notes, Ingeneria della Conoscenza e Sistemi Esperti 2002, Politecnico di Milano.

Lin L., Cao, L., Wang, J. and Zhang C. (2004) The Applications of Genetic Algorithms in Stock Market Data Mining Optimization, *Proceedings of Fifth International Conference on Data Mining, Text Mining and their Business Applications*, September 15-17, Malaga, Spain.

Schulenburg, S. and Ross, P. (2000) An Adaptive Based Economic Model, *Lecture Notes in Artificial Intelligence*, pp. 263-282.

Watkins, C.J.C.H. and Dayan, P. (1992) Technical Note Q-Learning, *Machine Learning*, 8, pp. 279-292

Wilson, S. (1995) Classifer Fitness Based on Accuracy, *Evolutionary Computation*, Vol. 3, No. 2, pp. 149-175.

Wilson, S. (1998) Generalization in the XCS Classifier System, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 665-674.

Zhou, Q.Q. and Purvis, M. (2004) A Market-based Rule Learning System, *Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, Vol. 32, pp. 175-180.