An Abstraction Algorithm for Genetics-based Reinforcement Learning

Will Browne
University of Reading
Berkshire
RG6 6AY, UK
+ 44 (0)118 378 6705
w.n.browne@rdg.ac.uk

Dan Scott
University of Reading
Berkshire
RG6 6AY, UK
+ 44 (0)118 966 6893
siu01ds@rdg.ac.uk

ABSTRACT
Abstraction is a higher order cognitive ability that facilitates the production of rules that are independent of their associations. Experience from real-world data-mining has shown the need for such higher level rules. The game of Connect 4 is both multistep and complex, so standard Q-learning and Learning Classifier Systems perform poorly. The introduction of a novel Abstraction algorithm into an LCS is shown to improve performance in the evolution of playing strategies.

Categories and Subject Descriptors
F.2.2 Nonnumerical Algorithms and Problems

General Terms
Algorithms, Performance.

Keywords
Learning Classifier Systems, Genetics-Based Machine Learning, Abstraction.

1. INTRODUCTION
During the application of the Genetics-Based Machine Learning technique of Learning Classifier Systems (LCS) to data-mine rules in the steel industry, Browne noted that many rules had similar patterns [1]. For example, there were many rules of the type 'if side guide setting < width, then poor quality product' due to different product widths. This resulted in a rule-base that was unnecessarily hard to interpret and slow to learn. A method is sought to generate higher order (abstracted) rules from the learnt base rules (see figure 1).

Abstraction may be defined as 'The act or process of separating in thought, of considering a thing independently of its associations' [2]. The implications of abstraction on learning have been considered theoretically for artificial intelligence and cognitive psychology [3]. Practical considerations relating to Reinforcement Learning have focused on Temporal Abstraction, often utilizing decision processes [4]. Here discrete abstraction for data-mining applications, using genetic-based processes, is considered.

GECCO'05, June 25-29, 2005, Washington, DC, USA.

The process of abstraction can be likened to Information Processing Theory [5] (a branch of Learning Theory), which suggests that humans have the ability to recognize patterns in data and chunk these patterns into meaningful units. The individual patterns do not necessarily remain in a memory store due to the holistic nature of the individual patterns. However, the chunks of meaningful information remain, and become a basic element of all subsequent analyses.

The Abstraction algorithm needs to perform this "chunking", for the individual patterns created by a learning system. The learning system selected was the XCS implementation of the Learning Classifier System concept as it has been shown to produce accurate and maximally general rule sets [6]. The LCS concept was derived from work by Holland [7] on developing artificial central nervous [cognitive] systems. Much past work has focused on improving learning performance, but recent work has revisited its cognitive abilities [8]. LCS use evolutionary computation to produce maximally general compact production rules, thus are a suitable technique to form base rules.

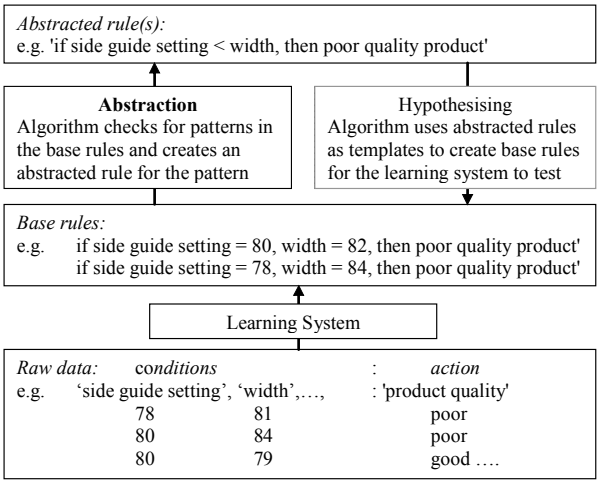

Figure 1 - Abstraction from data to higher order rules.

The first step in developing the Abstraction algorithm is to select a suitable test domain. Games, such as Chess or Connect 4, offer a useful environment in which to test an algorithm's ability to find

patterns within the data set and, more importantly, perform abstractions. Games are useful as they offer an environment that is well studied (meaning performance is easy to gauge), competitive, turn based (meaning time is not an important factor) and finite (although very large in most cases).

The goal of this project was to create an Abstraction algorithm that would generate rules for and play the game of Connect 4, due to this domain's scale, multi-step and non-deterministic properties.

2. BACKGROUND

Connect 4 is a turn based between two players, each trying to be the first to achieve four counters in a row (horizontally, vertically or diagonally). The game takes place on a 7 * 6 board; players take it in turns to drop one of their counters into one of the seven columns. The counters will drop to the lowest free space in the column. Play continues until the board is full or one player gets four in a row, see figure 2. Optimum strategies exist [9], so the problem is both known and bounded.

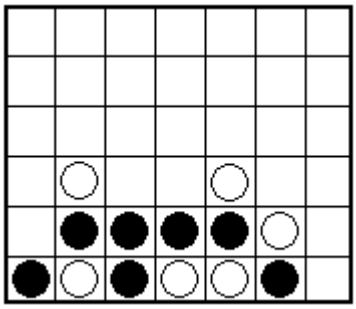

Figure 2 – Connect 4 board, black horizontal win

A client-server program of Connect 4 was written in Java, as Java Applets can easily be viewed on the internet, allowing a website to be constructed for this project [please visit: http://sip189a.rdg.ac.uk]. The client-server architecture of the game allows any combination of human/computer players to compete; meaning Co-Evolution between two different learning algorithms could be investigated using this program.

The client side of the program included seven expert systems, allowing human players to play against the computer as well as offering a benchmark against which to test various learning algorithms. The expert systems were hierarchical in nature, each one building on the last to become a more worthy adversary. The most basic level simply played the game randomly, whilst the most advanced mode had two moves look-ahead to play either winning or blocking moves. Human players have difficulty beating this mode unless they set up board states that give multiple win opportunities, see figure 3.

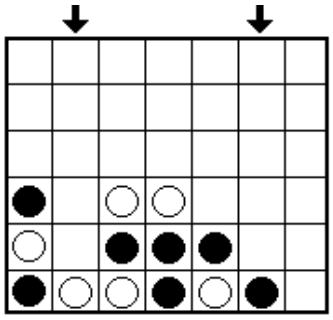

Figure 3 - Multiple win situation for black

Therefore, the most common of these situations were added to the expert system as 'filters' and image matching used - as analogous computer vision filtering - to improve performance. A filter, example shown in figure 4, is placed on a playable space and its score calculated by summing the associated squares (100 own counter, 0 space, -100 opponent's counter). The highest modulus score is played with a preference for positive moves for tied scores. Human players responded by using more disparate double win situations.

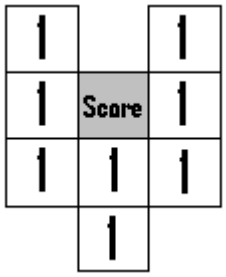

Figure 4 - Filter: Grey score box placed on playable space.

A Q-Learning [10] approach to the problem is implemented in order to provide benchmark learning performance. Two different approaches were taken to training the Q-Learning system. The first progressively trained the algorithm against increasingly hard opponents, whilst the second trained for the same number of games, but against the hardest opponent from the outset.

The Abstraction algorithm requires rules in order to perform abstraction. A well-known LCS, XCS [6], was implemented to create rules and provide a second benchmark learning performance.

3. LEARNING CLASSIFIER SYSTEM

This section outlines the architecture of XCS, including the required adjustments for the Connect 4 domain, so that it may train against the pre-coded expert system. A standard XCS [6, available from www-illigal.ge.uiuc.edu/] was implemented with the Abstraction algorithm (see section 6.2). Following these results tests were also conducted with a modified version of XCS (mXCS) that had its reinforcement learning component adjusted to complement the Abstraction algorithm (see section 6.3).

3.1 Setup and Board Representation

The board representation formed an important part of the LCS. Each space on the board could be one of three possible states, red, yellow or empty, however it was considered useful to further split

down the empty squares into two categories, playable and unplayable (unplayable squares are above the playable squares and become playable in the future as the game progresses).

A two character representation for each space was chosen, leading to an 84 character long string representing the board (running from top row to bottom row). The encoding for a red was chosen as "11" and a yellow was "10", a playable space was "00" whilst an unplayable was "01". Mutation may only generalize by replacing specific characters with a "#"; these hashes can stand for either a "1" or a "0". Therefore mutation can give rise to the following useful representations: "1#" means a space with a counter in (either red or yellow) and "0#" shows an empty space (either playable or unplayable) or simply "##" which could be anything. The mutation can also give rise to the following representations: "#0" either a playable or a yellow and "#1" either an unplayable or a red, however the usefulness of these mutations is questionable.

3.2 Gameplay and Rewards

LCS must decide upon the best move to play at its turn without knowing where its opponent will play in the subsequent turn. An untrained LCS will often play randomly as it attempts to learn the best moves to play. After each move has been played by the opponent, the LCS attempts to match the state of the board to its rules. Attached to each of these classifiers are three pieces of information: the move that should be played, the win score (the higher this is the more likely a win will occur) and the accuracy score (accuracy of the win score). Win scores of less than 50 indicate a predicted loss, greater than 50 is a projected win.

After matching, an action must be selected through explore, exploit or coverage. Exploring (which is most likely to happen) uses a weighted roulette wheel based on accuracy to choose a move. Exploiting chooses the move that has the greatest win score and is used for performance evaluation. Coverage generates a new rule by simply selecting a random move to play for the current board position.

The end of each game results in a win, loss or draw. In the event of a draw nothing happens and the game is simply reset. However, in either of the other two cases a reward/punishment system takes place. All the rules that were used in playing the game are rewarded/punished based upon their win score. If the game was won, the win score always gets increased by 2. If the game was lost, then the win score is always decreased by 2. The accuracy is altered based upon the current win score. For example, if the game is lost and the win score is below 50 (correctly suggesting a loss), the accuracy is increased by 2, whereas if the win score is above 50 (incorrectly suggesting a win), the accuracy is decreased by 2. In the LCS reward system the accuracy of losing rules is increased if they correctly label themselves as such. Eventually well fitted rules reach either 100 win score and 100 accuracy (absolute guaranteed victory) or 0 win score and 100 accuracy (absolute guaranteed loss).

3.3 LCS Genetic Algorithm

The LCS is trained for 100 games before the genetic algorithm (GA) is run. This allows rules the opportunity to be tested and to establish themselves.

3.3.1 Crossover

A two point crossover was used for this GA. Two rules were chosen randomly and the two crossover points were also chosen randomly. Large number of rules can be created that are invalid as the state that they represent could never occur on the board.

Originally the crossover was performed horizontally across the board as this was simplest to implement. However, this produced invalid states that contained 'floating counters'. Due to the rules of the game these floating counters cannot occur as the counter has to drop to the lowermost space they can occupy. Figure 5 demonstrates an example of a crossover between two perfectly valid states that leads to an invalid state.

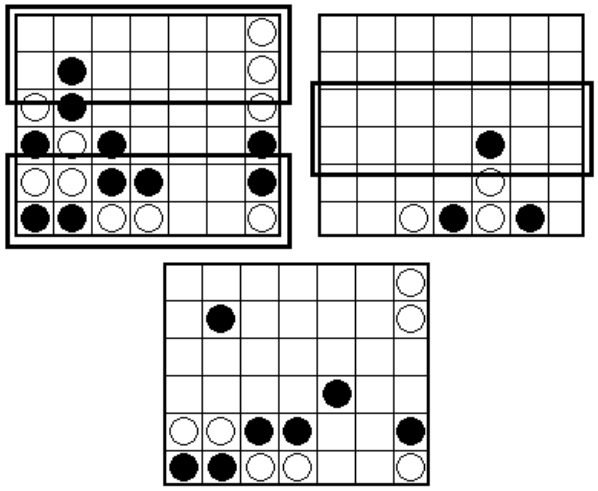

Figure 5 - Crossover leading to invalid state with floating counters

The crossover maybe performed vertically, using columns rather than rows (as shown by Figure 6). This eliminates the problem of "floating counters" but the problem of counter imbalance still occurs, where one color of counter out numbers the other. Due to the turn based nature of the game, a maximum imbalance of one more counter to the first player is possible. Imbalanced states are simply removed.

At the end of each generation (when the GA is run), 500 crossovers are performed in order to introduce new rules to the population. Due to the problem of counter imbalance occurring, not all of these 500 crossovers resulted in the insertion of a new rule into the population. This led to slight fluctuations in the population size, but had no effect of the performance of the LCS.

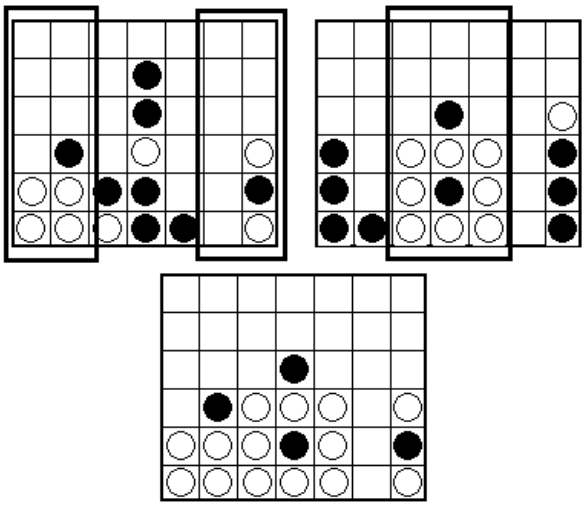

Figure 6 - Crossover leading to invalid state with counter imbalance

3.3.2 Mutation

The GA also includes a mutation algorithm that enables the LCS to perform generalisation. The mutation algorithm creates new states by factoring in #'s into already existing states. The chance of the mutation algorithm running on any one state was set to 1 in 100. The mutation algorithm causes far fewer invalidity problems than the crossover algorithm.

3.3.3 Duplications

If a rule is duplicated, then the duplicate is deleted and the numerosity [6] of the original rule is incremented. If a rule with a numerosity of greater than one is selected for deletion, it simply gets its numerosity decremented rather than the rule being deleted.

3.4 XCS features

XCS has a number of features that when implemented together distinguish it from other versions of the LCS concept. The important features include:

1. Fitness of rules based on accuracy of prediction, not strength.
2. Niche based rule discovering, instead of panmictic
3. Mutation generalizes only, specialization through coverage and crossover.
4. Deletion is panmictic, but considers action set size, which assists in negating the effects of an imbalance in class presentation rates from the data.
5. Michigan based, where the population consists of individual rules.
6. Designed to evolve a complete and accurate payoff map, not just the positive payoff state-action pairs.
7. Explore and exploit trials alternate.

The XCS was setup with the following parameters. N, the maximum population size was set at 5000. This was allowed to drift above 5000 due to coverage, but was always cut back to 5000 whenever the GA was run. Originally a population size of 1000 was chosen, however this gave rise to a far too competitive environment, with over half the population being deleted each time the GA was run, just to maintain the 1000 limit. With so many rules being deleted, 'good' rules did not get a chance to establish themselves. A learning rate, β, was not used in this XCS as, for reasons discussed in section 6.3, simple reinforcement learning was found to give the best performance.

θ_{GA} the GA threshold was set to 1000 games, the GA would run after a set of 1000 games had been played. χ, the crossover possibility was set to generate 500 random crossovers every time the GA is run. Of the 500 crossovers generated, approximately 100 in every GA run passed validity checks (see section 3.3) and were inputted into the new population. μ, the mutation rate was set at a 1% chance to receive a mutation and then a 2% that each character in that rule would receive a mutation. Deletion probabilities (θ_{del}) were based upon tournament selection of rule fitness and the number of rules deleted was chosen to keep the population size at 5000.

The standard reinforcement update for LCS is the Widrow-Hoff update [8], which is a recency weighted average. A Q-learning type update is used within the LCS technique for multistep decision problems [11].

4. ABSTRACTION ALGORITHM

The Abstraction algorithm was designed to work upon the rules generated by the LCS. Abstraction is independent of the data itself. Other methods, such as the standard coverage operator [7], depend directly on the data. Crossover and mutation depend indirectly on the data as they require the fitness of the hypothesized rules, which is dependent on the data. Abstraction is a higher order method, as once good rules have been discovered; it could function without the raw data being available.

The abstraction attempts to find patterns in the rules that performed best within the LCS. Having found a pattern common to two or more of the LCS rules, the Abstraction algorithm is to generate a new rule in the abstracted population based solely on this pattern. This allows the pattern to be matched when it occurs in any state, not just the specific rules that exist within the LCS.

4.1 Rule Selection

Not all of the rules generated by the LCS are worthwhile and therefore the Abstraction algorithm should not be run upon all of the rules within the LCS. The domain is noiseless, so the parameters chosen to govern the testing of rules for abstraction were the conditions that a rule must have a 100% win score and a 100% accuracy. Therefore the rules abstracted by the Abstraction algorithms should only be rules that lead to winning situations.

4.2 Windowing Function

The main mechanism that allowed the abstraction to perform was a windowing function that was used in rule generation as well as rule selection (when it came to choosing an abstracted rule to play). The windowing function acted as a filter that was passed over the 'good' rules generated by the LCS. This filter would compare two rules at a time for similarities that could lead to abstracted rules.

The windowing function worked in all directions on the board, horizontally, vertically and in both diagonal directions. The window size was set to 4 space/counters (8 characters in terms of the board representation). However code allowed for a window size of between 4 and 6 spaces/counter (8 – 12 characters in terms of the board representation), any greater than a window size of 6 and the vertical and diagonal windows no longer fit on the board.

4.3 Abstracted Rule Generation and Limitations

Any match that is found is turned into an abstracted rule, each rule had 8 characters (assuming a window size of 4) to represent the pattern occurring on the board. Each rule also had to be assigned a move to play whenever that rule was used. The move assigned was always chosen from one of the playable spaces within the pattern. An example rule is '10,10,10,00:11', which translate to 'if three red counters in a row and payable space in the next position, then play in the next position'. All rules entered the abstracted population with a win and accuracy of 50.

Several limitations were placed upon what was considered a valid match for the Abstraction algorithm, including ignoring all unplayable areas. A valid pattern had to contain at least one playable space and no more than 2 playable spaces. Patterns without a playable space are useless because rules as they offer nowhere for a move to be played. The second limitation placed upon the abstraction process was that a valid rule could have a maximum of one unplayable space. This helps limit the generation of "empty" rules. Figure 7 shows an example of two windowing functions finding a match and generating an abstracted rule.

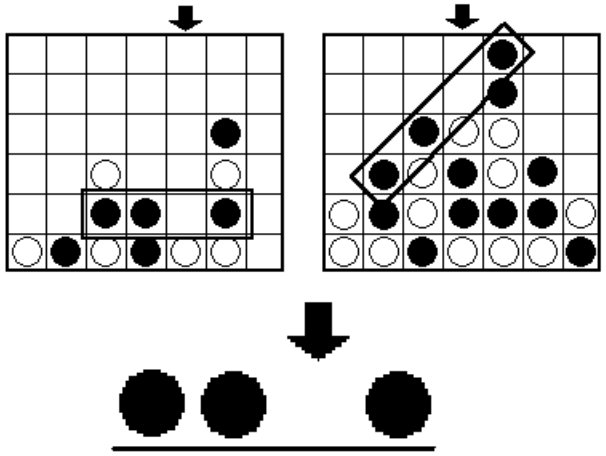

Figure 7 - Example of Abstraction Algorithms generating a new rule.

4.4 Abstraction Genetic Algorithm

As with the LCS, the Abstraction algorithm also had a GA that was run upon the population to generate new rules. It had a single point crossover and mutation; however it had no deletion algorithm as all the abstraction rules were kept. Duplication was prevented through a duplication check that was made each time a rule was to be inserted into the rule-base, including those generated by crossover and mutation.

5. LCS WITH ABSTRACTION

A LCS can function alone, but the Abstraction algorithm cannot function without a rule-base to work on; hence it needs an LCS to function alongside it. How the two are combined and work together is detailed in this section.

When the LCS with abstraction needs to play a move, the system searches the board for any matches within its abstracted rule set. The board is searched by passing the windowing function over the board (horizontally vertically and diagonally). A rule is then chosen out of all matched rules. When *exploiting* the rule with the best win score is chosen, whilst when *exploring* a roulette wheel based upon accuracy is used.

The chosen abstracted rule also has a move associated with it, however unlike the LCS rules the move does not relate directly to the board. With a window size of 4 counters the rule could occur anywhere on the board, horizontally, vertically or diagonally. Therefore an extra calculation is required to translate the abstracted rules' move into the corresponding move on the actual board.

If no abstracted rule is found after the initial search of the board state, then control of playing the move is handed to the LCS that chooses a move in the same way as described above in Section 3.2.

6. RESULTS

The following section details the results found during the trials of the LCS and Abstraction algorithm. Initial trials investigated the difficulty of the problem domain with standard Q-learning and XCS techniques. Preliminary tests of the Abstraction algorithm with XCS were followed by tests of the Abstraction algorithm with a modified XCS (mXCS) where the reinforcement learning complemented the abstraction. The use of abstraction as the training progressed was investigated. During these tests, each system was trained for 20,000 games against an opponent that played randomly. Finally, the robustness of the Abstraction algorithm to changes in the domain was tested by increasing the difficulty of the opponent.

6.1 Q-Learning and standard XCS

The Q-Learning Algorithm performed well in the initial 20,000 games (see figure 8), achieving an average win percentage of 69%. However, there was no progress in the wins as the 20,000 games progressed, with the win percentage always remaining at around 69%. This exhaustive search nature of the algorithm meant it took several weeks of computation on a 3GHz PC. Ideally, each test would have been repeated 10 times and the average results taken, but this was impractical due to time constraints.

The XCS performance trend was similar, with an average win percentage of 62% reached quickly, but no further improvements. Analysis of the rules showed that they had become trapped in local optima. A few specific strategies had been learnt, such as initially trying to build a column of counters in a given column. However, if this column happened to be blocked, then the overall strategy failed.

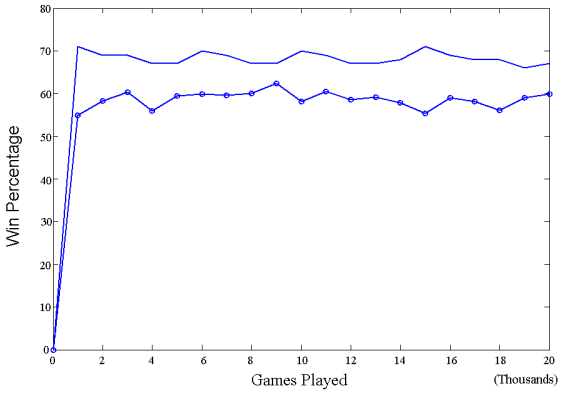

Figure 8 - Graphs of win percentages for the basic algorithms Solid Line - Q-Learning Algorithm, circle- Standard XCS

6.2 Standard XCS, plus Abstraction

When the Abstraction algorithm was included in the standard XCS the performance did not improve, see figure 9. This was because the XCS did not find sufficiently accurate rules for the Abstraction algorithm to be triggered.

The random nature of the opponent meant that a sometimes good strategy, such as build a column of counters in row one, was occasionally blocked. The prediction is updated by the Widrow-Hoff delta rule, which severely penalizes an incorrect prediction. Thus the accuracy of prediction never reaches a high stable level. Adjusting the learning rate (β) within the range 0.1-0.6 did not improve the performance. Reducing the threshold of what the Abstract algorithm considered as accurate to 85% accuracy of prediction also did not improve performance.

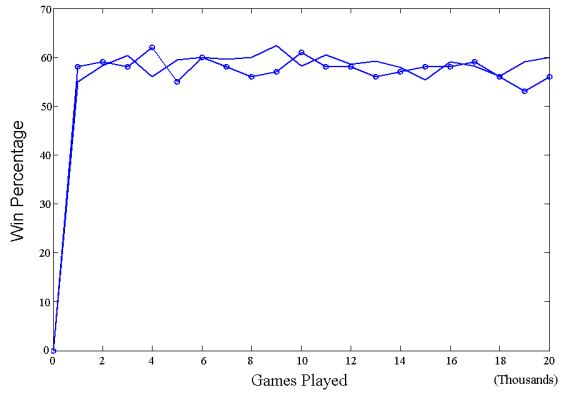

Figure 9 - Graphs of win percentages: Solid Line – XCS β = 0.2, circle- XCS with abstraction β = 0.2.

6.3 mXCS with and without Abstraction

The Widrow-Hoff delta rule was replaced by a simple reinforcement learning update, as outlined in section 3.2, where recency had a much reduced effect. This enabled mXCS to produce rules considered accurate enough to be abstracted. Although the rules were similar to those produced by XCS, the learning was more gradual, which prevented good rules from being replaced due to low accuracy of prediction.

Figure 10 shows the mXCS algorithm without abstraction, which reaches a similar performance level to standard XCS after 20,000 trials. Instead of taking under 2000 iterations to reach the final level, it takes over 6000 trials (see figure 11).

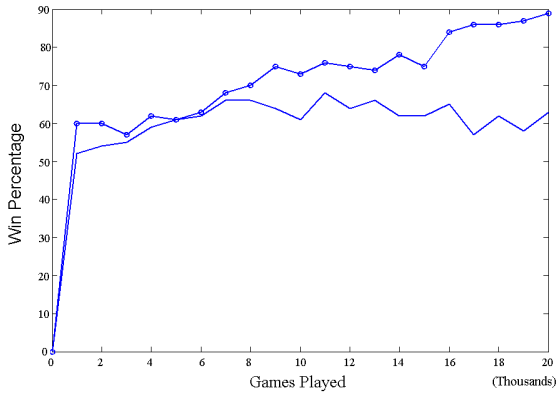

Figure 10 - Graphs of win percentages for mXCS: Solid Line - mXCS Algorithm, circle - mXCS with Abstraction.

When the Abstraction algorithm is added it produces a similar trend until 6000 trials. A significant improvement is noted after 8000 trials as the performance increases to 90%. This compares favorably with both Q-learning (69%) and standard XCS (62%), see figure 11.

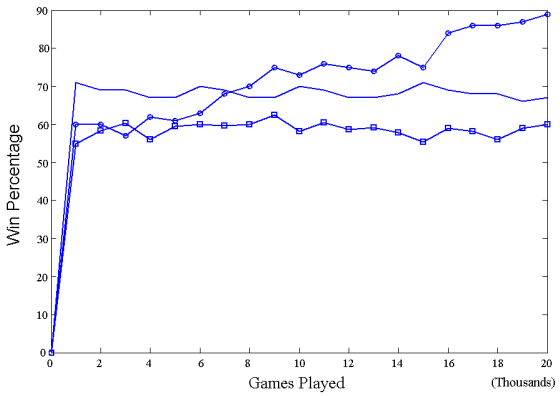

Figure 11 - Graphs of win percentages for the 3 algorithms Solid Line - Q-Learning Algorithm, square - XCS Algorithm, circle - mXCS with Abstraction.

During testing the rules that the Abstraction algorithm produced were observed and an interesting pattern arose in the order in which the abstractions were discovered. In early generations no abstracted rules are found, whilst mXCS attempts to establish a set of good rules that have a win and accuracy of 100. The first abstracted rules found are not rules for a direct win (i.e. 3 in a row and play in the fourth). The first rules that emerge are those rules that cause a 3 in a row situation with an empty playable fourth space.

Learning to form 3 in a row followed by learning to form 4 in a row is a novel example of incremental learning. Intuitively, it could be expected that learning to form 4 in a row, which is closer to obtaining the reward, would be achieved first. Incremental learning is hypothesized to be an important cognitive ability [8].

Whilst there is no direct feedback from the abstraction rule-base to the mXCS rule-base, it is possible to see them evolve together and there is a definite dependency between the two. With the introduction of abstracted rules to make 3 in a row, this is likely to occur far more often (as abstracted rules take preference over mXCS rules). With 3 in a row occurring more often, mXCS has more opportunities to conceive of rules that directly give a win. Therefore with more winning rules the Abstraction algorithm is more likely to come up with abstracted rules that lead to a direct win, greatly bolstering the winning ability of the algorithm.

6.4 Effect of Abstraction

The use of abstracted rules as training progresses can be monitored, see figure 12. As outlined in section 5, the combined system always plays a matching abstracted rule in preference to a matching base rule. After 8000 trials the base rules were accurate enough to allow abstraction to start. Once abstraction had started, the performance of the system continued to improve beyond that of standard XCS and Q-learning (see figure 11). A further 8000 trials occur where the system uses a combination of both base and abstracted rules. After this period the system just uses abstracted rules in its decision-making. Small improvements in performance occurred due to the action of the genetic algorithm in the abstracted population.

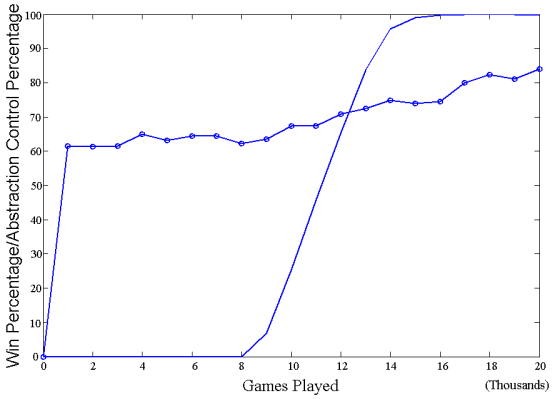

Figure 12 - Graph of percentage base rules versus abstracted rules (solid line) as training progresses (circle line).

The random opponent still defeats the system in 10% of the games when it chances upon a good strategy. As there are multiple positions for good strategy is to occur in, the system is rarely presented with them, which makes them difficult to learn. In order to determine the robustness and scalability of the techniques the difficulty of the opponent was increased.

6.5 Robustness of the systems

The opponent could now block a potentially winning three in a row state. The system has to learn to create multiple win situations, see section 2. This is a significantly harder problem, especially as the opponent could win either randomly or in the act of blocking, which halts the game.

Figure 13 shows that all algorithms perform poorly as all win percentages are under 20%. If no good base rules are found, then the Abstraction algorithm will not start.

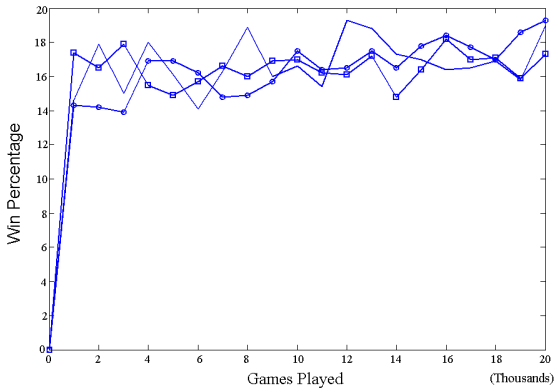

Figure 13 - Win percentages against blocking opponent. Solid Line - Q-Learning Algorithm, square - XCS, circle - mXCS with Abstraction.

Instead of training from the start with the harder opponent, it was decided to train first with the simple opponent and then switch to the harder opponent, see figure 14. After the switch, standard XCS performed better than the Q-Learning Algorithm, achieving a win percentage of 15%, it should be noted that the performance was less than the Q-Learning algorithm during the first 20000 games. Analysis of the Q-Learning algorithm testing showed that progressive training, from the easiest to the hardest opponent, caused it to get stuck in a local optimum with a win percentage of only 11%. The generality and adaptability of the standard XCS algorithm enables it to switch opponent without penalty.

The performance of the Abstraction algorithm was significant. Not only did it outperform standard XCS and Q-learning (53%, compared with 15% and 11% respectively), but it performed significantly better then when it had been trained only on the harder opponent (53% compared with 19%). This is a good example of incremental learning, where it is necessary to build up the complexity of the problem domain.

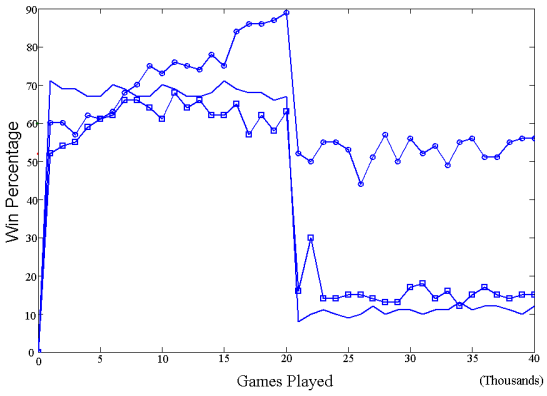

Figure 14 - Change in opponent at $20x10^3$ games played (Solid Line - Q-Learning Algorithm, square - XCS, circle - mXCS with Abstraction).

7. DISCUSSION

Abstraction may appear a trivial task for humans and the positive results from this work intuitive, but abstraction has not been routinely used in genetics-based reinforcement learning.

One reason is that the time that each iteration requires is an important consideration and abstraction increases the time for each iteration. Typically XCS takes 20 minutes to play 1000 games (and remains constant), mXCS with abstraction takes 20 minutes for 100 games (although this can vary greatly depending on the choice of parameters) and the Q-Learning algorithm ranges from 5 minutes for 1000 games initially and 90 minutes for 1000 games after 100,000 games training. However, given a fixed amount of time to train all three algorithms mXCS with abstraction would perform the best, once the initial base rules were found.

The Q-Learning algorithm has to visit every single state at least once in order to come up with a successful playing strategy. Whilst the Q-Learning system would ultimately play a very good game, weeks of computation failed to achieve the level of success the Abstraction algorithm had in a very short space of time (hours rather than weeks). Although better Q-learning algorithms (including generalization capabilities) exist [12] this choice of benchmark algorithm showed the scale of the problem, which is difficult to calculate.

The improvement in abstraction performance from standard XCS to the modified XCS was due to using simpler reinforcement learning. The Widrow-Hoff delta rule converges much faster, which for simpler domains that can be solved easily is beneficial. However, slower and more graceful learning may be required in complex domains when interacting with higher level features.

The abstracted rules allow the system to play on states as a whole, including those that have not been encountered, where these states contain a known pattern. This is useful in data-mining, but with the inherent dangers of interpolation and extrapolation. The abstracted rule-base is also compact as an abstracted rule covers more states than either a generalized LCS rule or a Q-learning state. Unique states may still be covered by the base rules.

Abstraction has been shown to give an improvement in a complex, but structured domain. It is anticipated that the Abstraction algorithm would be suited to other domains containing repeated patterns.

8. FUTURE WORK

Instead of the current linear filters in the Abstraction algorithm, it is possible to vary the size and shape in order to represent and hopefully discover advantageous multi-win situations.

The abstraction method is static and determined *a priori*, which is successful for this structured domain. The next stage is to evolve the abstracted rules and/or filters thus reducing the searching time.

A process termed 'hypothesizing' is proposed (see figure 1) where the abstracted rules form a template in order to produce new rules for the base population, with the worth of the abstracted rule being determined by the success of their hypothesized rules.

9. CONCLUSION

A novel Abstraction algorithm has been developed to successfully improve the performance of a genetics-based machine learning technique in a complex multi-step problem. It is hoped that this algorithm will help to fulfill the intended use of the LCS technique as a test bed for artificial cognitive processes.

10. ACKNOWLEDGMENTS

Our thanks to the Nuffield Foundation for their support through grant NUF-URB04.

11. REFERENCES

[1] Browne, W. N. L., The development of an industrial learning classifier system for data-mining in a steel hot strip mill. In Bull, L. *Applications of Learning Classifier Systems.* Springer, Berlin, 2004, 223-259.

[2] Oxford English dictionary, http://www.oed.com/.

[3] Thornton. C., Quantitative abstraction theory. In *Artificial intelligence and the simulation of behaviour. 1*, 3 (2003) 281-290.

[4] Sutton, R. S., Precup, D., and Singh, S. *Between MDPs and Semi-MDPs:* a framework for temporal abstraction in reinforcement learning. *Artificial intelligence*. 112, 1-2 (1999), 181-211.

[5] Miller, G.A. *The magical number seven, plus or minus two; Some limits on our capacity for processing information.* Psychological Review, *63*, (1956), 81-97.

[6] Butz, M., and Wilson, S. W. An algorithmic description of XCS. In *Soft Computing: a fusion of foundations, methodologies and applications, 6* (2002), 162-170.

[7] Holland, J. H. *adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan press, 1975.

[8] Butz, M., *Rule-base evolutionary online learning systems: learning bounds, classification and prediction*. PhD thesis University of Illinois, Illinois, 2004.

[9] Allis, V. *A Knowledge Based Approach of Connect 4.* Masters Thesis, Vrije Universiteit, Netherlands, 1988.

[10] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[11] Lanzi, P-L., Learning classifier systems from a reinforcement learning perspective. In *Soft Computing: a fusion of foundations, methodologies and applications, 6* (2002), 162-170.

[12] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press, 1998.