Learning to Control Dynamic Systems with Automatic Quantization

Charles X. Ling
Department of Computer Science
University of Western Ontario
London, Ontario, Canada N6A 5B7
Email: ling@csd.uwo.ca

Ralph Buchal
Department of Mechanical Engineering
University of Western Ontario
London, Ontario, Canada N6A 5B7.

To appear in *Adaptive Behavour*, 1994. (Not the final version; comments are welcome).

Abstract

Learning to control dynamic systems with unknown models is a challenging research problem. However, most previous work that learns *qualitative* control rules does not construct qualitative states — a proper partition of continuous state variables has to be designed by human users and given to the learning programs. We design a new learning method that learns appropriate qualitative state representation and the control rules simultaneously. Our method can aggressively partition the continuous state variables into finer, discrete ranges, until control rules based on these ranges are learned. As a case study, we apply our method to the benchmark control problem of cart-pole balancing (also known as the inverted pendulum). Experimental results show that not only does our method derive different partitions for the cart-pole systems with different parameters, but it also learns to control the systems for an extended period of time from random initial positions.

Key Words: Adaptive control, Reinforcement learning, Automatic quantization.

Short title: Learning to Control with Automatic Quantization.

1 Introduction

Controlling a dynamic system to achieve a certain goal involves receiving inputs from the system, making decisions, and applying actions to the system. If the model of the system is known or is relatively simple, traditional control methods (such as system identification) can identify the model and derive control rules from it. However, if many aspects of the systems are *unknown* (such as temperature control in a building, economics, weather, and so on), or the systems are too complicated, adaptive control methods that *learn* control rules directly (i.e., without identifying the model) through trials have been shown to be promising and effective (cf. (Michie & Chambers, 1968; Barto, Sutton, & Anderson, 1983; Anderson, 1986; Sutton, 1988; Lin, 1990; Urbancic & Bratko, 1993)). Although this is a typical adaptive control problem, it can be regarded as an animat problem as well[1] (Meyer & Guillot, 1994). That is, an animat (the controller) tries to survive (achieving a certain goal) in an unknown or hostile environment (unknown dynamic system model) by learning (adapting) and interacting with the environment (receiving inputs from and applying actions to the dynamic system).

In contrast to the supervised learning with classified training examples, information on the correctness of actions of the controller usually is not available. The learner (or animat) only receives a weak feedback (performance measure) indicating a *delayed* and *cumulative* effect of the actions. Reinforcement learning is a powerful adaptive control method for learning with weak feedback. It is based upon the commonsense idea that if an action in a sequential decision task is followed by an improvement in the state of affairs (determined by the performance measure), then the tendency to produce that action is strengthened, i.e., reinforced. The central issue of reinforcement learning is credit assignment that credits or penalizes the series of actions that the controller performs.

Many direct, adaptive control methods learn *qualitative* control rules (or strategies) that map from the qualitative states to actions. Although state variables that describe dynamic systems usually are continuous, qualitative control strategies have several advantages over quantitative ones (such as a set of numerical equations). First, with qualitative control strategies, decision making is essentially a table look-up, and thus is very fast and can be used on-line. Second, we can assign linguistic terms (such as *small, medium, large*) to the partitions of the qualitative state variables and produce production rules that may

[1]An animat is a simulated animal or a real robot.

be comprehensible to humans. This is an important aspect in cognitive modeling (cf. (Ling & Marinov, 1993, 1994)); the animat may want to communicate the learned strategy in the form of qualitative rules to humans or to other animats. Numerical controllers, such as neural networks, can be blackboxes that defy analysis. Third, for many control problems (such as the cart-pole balancing problem that will be discussed in this paper), the number of qualitative states required is not very large, and thus, learning qualitative control rules can be simpler than learning quantitative ones.

The format of qualitative control rules can be represented as

IF the system is in a qualitative state S THEN apply action A.

However, most previous work that learns qualitative control rules does not construct qualitative states automatically — a proper partition of the continuous state variables has to be designed by humans and given to the learning programs[2]. We present a new learning method that learns appropriate qualitative state representation and the control rules *simultaneously*. That is, the animat not only has to develop his own qualitative states for a particular environment, but also control rules based on them.

Thus, the learning process involves two tasks: one is to partition or quantize the continuous state-space into appropriate discrete regions, and the other is to learn the control rules for these regions. These two tasks interact with each other, and thus make the whole learning task very difficult. Our method, called STAQ (Set Training with Automatic Quantization), can aggressively partition the continuous variables from coarse to finer, discrete ranges, and learn control rules based on these newly developed ranges. This allows the animat to transfer from quantitative reasoning to qualitative reasoning, from sensory representation to symbolic representation, and from a coarse symbolic representation to a better symbolic representation, through trial-and-error and interactive problem solving. As a case study, we apply STAQ to the benchmark, non-linear system of the cart-pole balancing (also known as the inverted pendulum) problem. Experimental results show that not only does STAQ derive different discrete state representations for the cart-pole systems with different system parameters, but also learns to control the systems for an extended period of time from random initial positions.

The paper is organized as follows. In Section 2 we discuss the general strategy of

[2]The problem of discretizing continuous attributes in *supervised* learning has been studied extensively (cf (Schlimmer, 1987; Quinlan, 1986; Utgoff, 1986; Matheus & Rendell, 1989)). However, these methods generally are not applicable in reinforcement learning due to the lack of direct feedback (i.e., classification).

reinforcement learning with automatic quantization. In Section 3 we review the cart-pole balancing problem and previous work. In Section 4 we apply our STAQ algorithm on the cart-pole balancing problem, and show experiment results.

2 Learning with Automatic Quantization

The basic idea of learning with automatic quantization is to start with a very coarse partition of the continuous state variables, and gradually refine it to finer partitions. To learn appropriate qualitative state representation and the control rules simultaneously, two procedures are iterated repeatedly. Briefly speaking, the first procedure is a reinforcement learning algorithm that tries to learn the best control rules based on the current partition via trial and error. During reinforcement learning, evaluation of the regions that may require further partitioning is also carried out. The second procedure is a quantization process that refines the current state representation. These two processes iterate until enough resolution is reached in the second procedure such that good control rules are learned in the first procedure.

Any non-overlap partition with n state variables forms *boxes* in an n dimensional space. Learning qualitative control rules can be regarded as setting appropriate actions for all boxes. (We consider only binary actions in this paper.) Therefore, the first procedure is a reinforcement learning algorithm that determines the best settings of the actions in all boxes for achieving the goal. The process utilizes a credit assignment policy that penalizes boxes whose current actions result in performances well below average. Learning is a process that updates and chooses better actions for these boxes. To evaluate the need for further partitioning in a particular box, a variable called the *flip count* (or FC for short) is carried in each box. When the action in a box is changed during reinforcement learning, the FC value of that box is increased. The rationale is that the FC value accumulated during the reinforcement learning represents the number of "mind changes" that have occurred for that box. The box with the highest FC value is the most *inconsistent* one since it is penalized no matter what action it takes. Therefore, it represents the portion of the state space region that needs further partitioning. Note that only when the current action performs well below average (see Section 4.1 for details) does it have a more chance of being changed, and therefore, two equally good actions in a box will not result in flipping of the action often in that box.

When the current resolution of the state space is not fine enough, the goal cannot be reached no matter how actions in boxes are set. Therefore, a *halting criterion* is needed to stop reinforcement learning (the first procedure), and move to the quantization process (the second procedure).

The quantization procedure needs to decide which variable at which range needs further partitioning. Two strategies of partitioning are possible. One strategy is to partition further *within* the box with the highest FC value, the other is to partition the whole "slice" of boxes that passes through the box with the highest FC value into two "slices". The set of boxes that have a *fixed* range of one variable while all other variables vary is called a *slice*. Although the first strategy partitions only at the local region where partition is mostly needed (such as (Moore, 1994)), it produces non-uniform state representation; there are boxes within boxes. The second strategy produces *uniform* partition of each variable in the whole state space, thus it is more natural to assign linguistic terms (such as *small, medium, large*) to the partitions of the state variables for production rules. In this paper, we focus mainly on the second strategy, which splits the whole slice of boxes into two.

Starting with a very coarse partition, these two processes iterate until enough resolution is reached in the second procedure such that good control rules are learned in the first procedure. The general strategy of learning control rules with automatic quantization is outlined in Table 1. We describe each component of the algorithm for the cart-pole balancing problem in detail in Sections 4.

3 The Pole-Balancing Problem

Cart-pole balancing (or inverted pendulum) is a typical non-linear dynamic system. Learning to balance the cart-pole was the subject of one of the earliest experiments conducted in machine learning, and has been a benchmark test for control methods. We review the cart-pole balancing problem and previous learning methods of the problem in this section.

```
loop
    repeat
        Call Procedure 1
        /* Learn appropriate actions and update the FC values in boxes */
    until (The goal is achieved) or (Halting Criterion = true)
    if the goal is achieved
        then Succeed and exit
        else Call Procedure 2
                /* Split the slice of boxes with the highest sum of FC values */
```

Table 1: The algorithm of learning control rules with automatic quantization

3.1 The Problem

The cart-pole balancing problem can be illustrated in Figure 1. A rigid pole is hinged to a cart, which is free to move within the limits of a track. The learning system attempts to keep the pole balanced and the cart within its limits by applying a force of fixed magnitude to the cart, either to the left or to the right (i.e., binary actions).

The *goal* is to balance the cart-pole for as long as possible. Failure is reported when the angle of the pole or the position of the cart exceeds certain limits defined by users. In the cart-pole system, we define that the control strategy fails to balance the cart-pole if $|x| > 2.4$ meters (the cart bumps against the ends of the track) or $|\theta| \geq 12°$. All non-failure situations are treated equally; i.e., there is no graded feedback for non-failure states. Thus, stability, or the magnitude of the off-center errors, is not included as part of the goal. The failure signal is the *only* feedback received by the learning program. This means that the learning program is given a very weak guidance — a bad action may cause failure long after the action is applied.

The sensory inputs (continuous state variables) to the learning program at any instant of time is a vector of four real number parameters:

x : the position of the cart on the track

$\dot{x}$: the velocity of the cart

θ : the angular position of the pole

$\dot{\theta}$: the angular velocity of the pole

These state variables describe the dynamic system completely for the purpose of the cart-pole balancing. Since the cart has a mass, applying an instantaneous force does not produce an instantaneous change in position and velocity. In addition, since the track has a limited length, "over-shooting" (pushing the cart towards the end of the track) is sometimes necessary to allow the cart to return the center of the track. Learning to balance both cart and pole without a model clearly is not a trivial task.

We used a simulator to mimic the actual cart-pole system to interact with the learning program. The simulator is modelled by two non-linear second order differential equations that accurately approximate the real physical system (Cannon, 1967).

$$\ddot{\theta} = \frac{G \sin\theta - \cos\theta (F + M_p L \dot{\theta}^2 \sin\theta)/M_t}{L(4/3 - M_p \cos^2\theta / M_t)}$$

$$\ddot{x} = \frac{F + M_p L(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{M_t}$$

where

Pole mass (M_p) = 0.1 kg.

Cart mass (M_c) = 0.9 kg.

Total mass (M_t) = 1.0 kg.

Pole length ($2L$) = 1 meter.

Applied Force (F) = $\pm$ 10 N (left or right).

Given the values of the state variables θ_i, x_i, $\dot{\theta}_i$, $\dot{x}_i$, and F at the i^{th} sampling, the values of the state variables at the next sampling are calculated by:

$$\begin{aligned} x_{i+1} &= x_i + T\dot{x}_i \\ \dot{x_{i+1}} &= \dot{x}_i + T\ddot{x} \\ \theta_{i+1} &= \theta_i + T\dot{\theta}_i \\ \dot{\theta_{i+1}} &= \dot{\theta}_i + T\ddot{\theta} \end{aligned}$$

where sampling period $T = 0.02$ sec.

The learning algorithm is model free: it uses the simulator merely as a blackbox and refrains from using any domain knowledge embedded in the simulator, not even the symmetric property. The *only* information that it receives is vectors of these four continuous state variables $(x, \dot{x}, \theta, \dot{\theta})$ at every sampling, and the failure signal when the system fails.

3.2 Previous Work

The pole-balancing problem was first studied by Widrow and Smith (1964), and by Michie and Chambers (1968), and has become a benchmark task for studying control methods of non-linear systems (cf. (Anderson & Miller, 1990) for literatures before 1990). However, most previous work which learns the qualitative control rules requires partitioned state variables.

3.2.1 BOXES

Michie and Chambers (1968) design and implement an algorithm called BOXES as a reinforcement learning algorithm for the cart-pole balancing problem. However, quantization is not part of the learning task. Each of the four continuous state variables $(x, \dot{x}, \theta, \dot{\theta})$ are partitioned into 5, 3, 5, and 3 ranges respectively through analyses. This creates 225 boxes and at any instant the system is in one of these boxes. BOXES gathers information on how well each action performs during a trial, and updates actions with bad performance in boxes at the end of the trial when the system fails. As Michie and Chambers point out, the choice of ranges for the state variables is *critical* to the success of the BOXES program. Another weakness of BOXES is that the learned control strategy is not *robust* enough in the sense that a change in simulation parameters results in a significantly longer learning time (Sammut & Cribb, 1990). The acquired control strategy is not *generic* enough either, in the sense that it does not perform well in testing trials with random initial positions.

Some further work has been done following the BOXES algorithm. For example, Sammut and Cribb (1990) show that adjacent boxes learned may be merged to produce simpler rules that make more sense to humans. They also study a new algorithm called the *voting algorithm* which produces a more generic control strategy that can balance the cart-pole from any initial position within a specific range. The method first derives specialized control strategies for 20 specific initial positions via BOXES, and then merges them to a single controller via voting.

3.2.2 The AHC algorithm

The Adaptive Heuristic Control (or AHC for short) is a reinforcement algorithm developed by Sutton (Sutton, 1984). This algorithm takes a more incremental approach to learning the evaluation functions. Instead of updating actions (i.e., learning) at the end of the trial as in BOXES, it updates actions after each step during the trial. The central idea is to compute the error values by measuring the differences between temporally successive states. Like BOXES, AHC (Barto et al., 1983) also requires the state space to be partitioned into predefined ranges, and the learned control strategy is not generic for the random tests (Sammut & Cribb, 1990). Anderson (1989) implements AHC in a neural network that avoids partitioning the state variables. However, the outcome of the learning is a numerical relation embedded in the network.

The AHC algorithm is a special version of the *temporal-difference learning* (TD) (Sutton, 1988) — a general way of making long-term predictions based on incremental, immediate results. TD is basically what is used to handle delayed reward in all reinforcement learning methods. One can classify reinforcement learning into two major families[3]: one that includes AHC which is related to the policy iteration method of dynamic programming, and the other that includes Q-learning (Watkins, 1989) which is related to the value iteration method of dynamic programming. However, TD theorems generally only apply to cases with distinct-state table-lookup (i.e., completely partitioned state space). As pointed out by Sutton, Barto, and Williams (1991), "In large problems, or in problems with continuous state and action spaces which must be quantized, these methods become extremely complex." More recently, Chapman and Kaelbling (1991) design a so called G algorithm that starts with one state of the Q table, and gradually splits it into a tree-structured Q table. The program is based upon the assumption that bits that describe system status must be individually, rather than collectively, relevant. Mahadevan and Connell (1991) propose a dual approach in which they start with fully differentiated inputs (or overly refined boxes), and then merge them when they are found to behave similarly. The number of initial states in this approach can be very large. Moore (1994) designs the Parti-game algorithm which maintains a decision-tree partitioning of the state space and applies techniques from game-theory and computational geometry to adaptively develop higher resolution in critical areas in the state space. The efficiency is high and the results are impressive. However, the method assumes that the task is specified by a

[3]Private communication with Richard Sutton.

goal state, rather than by a reward function as the cart-pole system studied in this paper.

3.2.3 The CART program

The CART program (Connell & Utgoff, 1987) removes the assumption of pre-partitioning the state variables by using a different knowledge representation and a new action selection mechanism. The CART program learns a *numerical* equation that approximates the control surface by a means of interpolation. Some domain knowledge and heuristics for action selection are embedded in the the numerical equation that is computed after each move. The central idea of this approach is to search for a more desirable state. At each step, if repeating the previous move appears to take the system to a better state, the move is repeated. The program is shown to learn to balance the cart-pole system from one initial position after only 16 trials.

The CART program avoids the requirement of partitioning the state space. However, the control strategy learned is a very complicated numerical equation that is not comprehensible, and it cannot generate any explanation or rules. The learned control strategy is found not to be generic for the random testing trials (Sammut & Cribb, 1990). In addition, the learning time as well as the decision making time – just a table look-up in BOXES – is very large (hours), making it unsuitable for real-time applications.

3.2.4 Fuzzy Logic Control Systems

Berenji (1992) extends Anderson (1986)'s work on neural network controllers, and designs a neural fuzzy logic controller ARIC (Approximate Reasoning-based Intelligent Control) for the cart-pole balance problem. ARIC consists of two multilayer neural networks, AEN (action-state evaluation network) and ASN (action selection network). ASN corresponds to the fuzzy logic controller, in which each hidden unit represents a fuzzy rule. Modifying weights in ASN fine-tunes the membership functions used in the fuzzy rules. ARIC starts with a roughly correct set of fuzzy logic rules, and adjusts the slope and cut-off points of the fuzzy membership functions through reinforcement learning. The system starts with a correct partition and a set of 13 fuzzy control rules that was previously written (Berenji, Chen, Lee, Jang, & Murugesan, 1990). Berenji (1992) showed that with slight changes in the slope and shift of the monotonic (linear) membership functions of the conclusion of these rules, ARIC learns to control the cart-pole system within a few trials. There are two major differences between his work and ours: first, the partition of state variables is fuzzy in ARIC, and therefore boxes overlap each other; and second, his underlying learning mechanism is based upon multilayer neural networks.

4 The STAQ Algorithm

As we have seen, most previous approaches that learn qualitative control rules for the cart-pole system do not construct qualitative state representation. Our goal is to develop a learning algorithm that consists of:

- an automatic quantization process. The algorithm should be *robust* in the sense that a different qualitative state representation may be developed for the cart-pole system with different system parameters (such as with a longer pole).
- a reinforcement learning algorithm that learns *generic* control strategy in the sense that it should not just be able to balance the cart-pole from one initial position. Instead, it should be able to achieve good results in a variety of the initial positions. Learning to control the cart-pole system with random initial positions is much more difficult than with one initial position only (Berenji, 1992, page 285).

As we have discussed in Section 2, we need a reinforcement learning algorithm that produces generic control rules, a halting criterion that stops reinforcement learning when the resolution of the current state representation is not enough, and a partitioning algorithm that refines the state representation. These will be discussed in detail in Sections 4.1, 4.3, and 4.4 respectively.

4.1 The Set Training Algorithm

We choose the well-studied BOXES algorithm for the cart-pole system and extend it to a new algorithm called the *set training* algorithm. Learning in BOXES takes place at the end of the trial when the system fails from *one* initial position. It has been shown (Sammut & Cribb, 1990) that this does not produce generic control rules in testing trials with different initial positions. In the set training, however, learning is based on *sets* of training instances with different initial positions, and the performance of the control strategy is measured by the average balancing time of the trials in the training sets. The goal is to reach an average balancing time of 10,000 sampling steps (200 seconds) of the training instances in the sets.

The initial positions in the set of the training instances should be distributed evenly in a wide range of the state space. Experiments show that at least 20 training instances are needed. The 20 initial positions are randomly chosen within $|x| < 1.2\ m$ and $|\theta| < 6°$ (i.e., half of the full range) and $\dot{x}$ and $\dot{\theta}$ are both set to 0. Values close to the boundary ($x = \pm 2.4, \theta = \pm 12°$) are not included since most of them are "doomed" states in which failure is unavoidable.

Learning is a process that sets the proper actions of all boxes through trials. A *trial* is a process of balancing the cart-pole from a particular initial position using the current actions until the system fails. In BOXES, the learning phase takes place at the end of each trial; in the set training algorithm, learning takes place after 20 trials in the training set. During the 20 trials, "scores" are kept in all boxes of how well the actions perform. Very similar to the BOXES, each box keeps four local variables for the *life* and *usage* of the left and right actions respectively. These are left life (LL), left usage (LU), right life (RL), and right usage (RU). The *life* of an action (LL and RL) in each box is a weighted sum of the duration that the action can balance the cart-pole system without failure. The *usage* of an action (LU and RU) in each box is a weighted sum of the number of the corresponding action used. Initially, the variables in all the boxes are set to zero, and all actions are set to random. During each trial, if a box is entered N times, then the moments that the box is entered are denoted as $T_1, T_2, ..., T_N$, where T_i is the elapsed time number of sampling periods from the start of the trial to the given entry of the box. If T_F is the total life time of the current trial, then $(T_F - T_i)$ is the duration from the time when the box is entered to the time when the cart-pole fails. According to the definitions of life and usage above, the life and usage of the current action set in each box are updated at the end of each trial by:

$$\begin{aligned} life_{i+1} &= life_i\ DK + \sum_{i=1}^{N}(T_F - T_i); \\ usage_{i+1} &= usage_i\ DK + N; \end{aligned}$$

where DK is the decay factor, set to 0.98. The life and usage of the *unused* actions in all boxes are "inherited" from the previous values with a decay factor:

$$life_{i+1} = life_i\ DK;$$

$$usage_{i+1} \;=\; usage_i\; DK;$$

The major difference between set training and BOXES is that only at the end of 20 trials are the actions in boxes updated. The ratio of life/usage indicates the overall performance of the action. It is the average time, from start of the training, that the action can keep the cart-pole system alive. The action in each box should be changed if the ratio of the current action is well below the ratio of the other action in the same box. One might simply compare the ratios of left and right actions and choose the setting that has the higher ratio. However, this would be caught in a "local minimum": whenever an action starts to look good, its ratio is higher, so it is reinforced without making any attempt to try the other action. Therefore, the decision is made stochastically based on the comparison between the ratio of life over usage of one action and the ratio of global life (GL) over global usage (GU) of actions in all boxes. We used a new updating strategy which is shown to be more robust: if the current action of a particular box is left, then the PR (performance ratio) of this box is calculated by $PR = \frac{LL/LU}{GL/GU}$. A random number p between 0 and 1 is drawn, and if

$$\frac{1}{1+e^{k\;PR}} > p\ ,$$

then the action (left) in the box will be updated and set to the other action (right). A similar sigmoidal logistic distribution is also used in determining the change of actions in (Selfridge, Sutton, & Barto, 1985, page 672), and the constant k in the formula is set to 4.5 by trial and error. Therefore, the smaller the ratio PR, the more probably the action will be changed since the current action performs well below average. After updating, a new iteration starts with 20 new randomly selected trials of different initial positions in the training set. Through many such iterations, the ratios of life over usage increase gradually, the average balancing time of 20 initial positions in the sets also increases, and thus better control rules are learned.

4.2 Results of the Set Training Algorithm

We apply the set training algorithm on the cart-pole system with the partition given in (Selfridge et al., 1985) with 162 boxes. In a typical run, the maximum average balancing time of the 20 initial cases reaches 44,000 time steps[4] after 43 iterations, with a maximum

[4]Note that 44,000 time steps represent 14 minutes in real time.

individual balancing time greater than 100,000 steps[5]. With more iterations over 20 training instances, the average balancing time stabilizes to about 37,000 time steps. The learning curve of the set training is demonstrated in Figure 2.

We note that the learning curve in the early period of learning fluctuates substantially. This is because the learning program is informed of failure only when the cart-pole fails — long after bad actions are taken. The evaluation of actions, therefore, requires many trials to be accurate. In addition, boxes are related to each other in a complicated way; changing the action in one box may affect the whole control strategy dramatically. Therefore, like other reinforcement learning algorithms, the set training algorithm has a non-monotonic learning curve. As we can see, however, the performance does tend to be better and more stable with more iterations.

To test the genericness of the learned control rules (obtained at 43 runs), 100 random tests are drawn with random initial positions within $|x| < 1.2\ m$ and $|\theta| < 6°$ (i.e., half of the full range). In the typical run, 75 out of 100 random tests have over 10,000 time steps balancing time, with an average of over 37,000 time steps. This shows that the set training algorithm produces quite generic control rules. However, some "holes" with low balancing time still exist in the state space. Upon examination of the 20 training instances, 5 out of the 20 still have low balancing time. This is because the set training algorithm tries to increase the *average* balancing time of the 20 initial instances. Therefore, if some of them have very large balancing time, others with low balancing time may not be discovered. Future work is needed to produce more generic control rules. In addition, we found that balancing time in the space of initial positions is not smooth; the balancing time of the initial positions very close to the one that has a very long balancing time can be very short. That is, starting from two initial positions that are very close together, the trajectories of the control process in the boxes can be very different after a few seconds — a typical behavior in a chaotic system. See more discussions in Section 4.7.

4.3 Halting Criterion

After each iteration of 20 trials in the set training, STAQ needs to determine, via the *halting criterion*, whether more iterations are needed for the set training algorithm to

[5]An upper bound (100,000 steps) is set to prevent the set training going on forever. If the balancing time is over 100,000 steps, 100,000 is taken as the balancing time.

induce better control rules, or whether the resolution of the current state representation is insufficient, so that further partitioning is needed. Since the average balancing time fluctuates in the early period of learning, premature decisions should be avoided. Our halting criterion is a look-ahead procedure that tests if the current highest average balancing time is exceeded by the next c iterations. If all of the next c iterations in the set training produce a lower balancing time, then the halting criterion is true and set training halts. The following method is used to *derive* the proper values of c. We first set c to be a large number, and then observe the minimal value of c needed after each partition such that the maximum average balancing time would not be missed during the set training. For example, if the maximum average balancing time is reached at the 120th iteration, and the previous maximum is reached at the 90th iteration, then c should be set to at least 30 in order to capture the maximum at the 120th iteration. We find that c increases as the number of partitions increases. After applying the set training algorithm on various partitions, we obtain the best values of c for these partitions, and we then try fit these values using some simple equation. We find that a simple linear relation $c = 30 + 10i$ where i is the number of partitions fits the values of c obtained, and this linear relation is used in the halting criterion.

4.4 The Partitioning Algorithm

When the halting criterion is true and the goal has not been achieved, STAQ calls the the partitioning algorithm to refine the current state representation. As we have discussed in Section 2, a variable FC (flip count) is kept in each box. When the action in a box is changed (from left to right or right to left), the value of the FC is increased. When the set training stops due to the halting criterion, the box with the highest FC value represents the most *inconsistent* region in the state space that needs refinement. Since partitioning takes place across the whole state space rather than only within that box, we need to decide which slice that passes through the box with the maximum FC value is to be split into two. The strategy we use is quite simple: we compare the summation of the FC values of the four slices, and choose the slice with the highest summation. The chosen slice is then split into two at the middle point. The contents of new boxes are inherited from the parent boxes, while the contents of other boxes remain the same. Therefore, the previously learned control strategies are not forgotten and are carried over to the new, enlarged partitions.

The STAQ algorithm starts with a set of 16 boxes with 2 ranges for each of the four state variables split at the middle (zero point), since this would be the first split point for each variable.

4.5 Results of the STAQ

Using the standard parameters of the cart-pole described in Section 3, STAQ splits, in most simulations, 6 times before it learns a control strategy that that achieves the goal, producing an average balancing time over 10,000 sample steps. Figure 3 shows a typical learning curve after the 4th, 5th, and 6th splits, and Figure 4 is the expansion of Figure 3. As expected, before the 6th split the partitions do not provide enough resolution for good control rules, so the performance during the set training is very poor. After the 6th split however, enough resolution is achieved, and good control rules are learned such that the maximum average balancing time of 20 training instances reaches 21,000 time steps. The number of iterations for each successive partition is 46, 60, 68, 80, 83, 110, and 267 respectively (the last one passes the success criterion without partitioning). The average balancing time of 100 random tests with the initial positions $|x| < 1.2\ m$ and $|\theta| < 6°$ (i.e., half of the full range) is over 17,000 time steps. The learned control rules are quite generic since most of the 100 random tests produce reasonably long balancing time.

It is interesting to examine the partitions obtained by STAQ. Different runs of STAQ do sometimes result in different partitions, but the algorithm often needs only 6 partitions, producing fewer boxes (e.g., 144) than the original BOXES (225) and AHC (162). See Table 2 for an actual partition developed by STAQ.

State variables	**Partitions** (New ones inserted by STAQ are shown in bold)
x	$-2.40 \ldots 0.00 \ldots \mathbf{1.20} \ldots 2.40$
$\dot{x}$	$-6.00 \ldots \mathbf{-3.00} \ldots \mathbf{-1.50} \ldots 0.00 \ldots 6.00$
θ	$-0.20 \ldots 0.00 \ldots \mathbf{0.10} \ldots 0.20$
$\dot{\theta}$	$-4.00 \ldots \mathbf{-2.00} \ldots \mathbf{-1.00} \ldots 0.00 \ldots 4.00$

Table 2: Results of STAQ with the Standard Simulator

4.6 Changes of the Dynamic Systems

To evaluate STAQ's robustness, i.e., if it can develop different qualitative state representation for the dynamic systems with different parameters, the simulation parameters of the cart-pole are altered. Again, STAQ has no knowledge about these changes in the systems. Unlike Selfridge et al. (1985)'s work in which system parameters are altered to see how the learning program adapts to changes from the previously-learned controller, STAQ learns to control each "new" dynamic system from scratch.

4.6.1 Unequal Forces

The STAQ program is required to learn to control the system when the force applied to the cart-pole system is changed; the left force is reduced to 5 N and right force is maintained at 10 N. This can be viewed as an inaccurate approximation of balancing the cart-pole on an inclined track, and it is a more much difficult learning and control problem. Indeed, STAQ requires more splits, often 9 splits (resulting in 288 boxes), in order to cope with the new and more difficult dynamic system. In one typical run, STAQ reached 52,000 time periods as the maximum average balancing time of the 20 training instances. The number of iterations for each successive partition is 74, 52, 60, 70, 82, 90, 136, 171, 225, and 189 respectively. The average balancing time of 100 random tests is 19,000. Table 3 shows that the partition obtained for the new system.

State variables	**Partitions** (New ones inserted by STAQ are shown in bold)
x	$-2.40 \ldots \mathbf{-2.10} \ldots \mathbf{-1.80} \ldots \mathbf{-1.20} \ldots 0.00 \ldots \mathbf{1.20} \ldots 2.40$
$\dot{x}$	$-6.00 \ldots \mathbf{-3.00} \ldots 0.00 \ldots 6.00$
θ	$-0.20 \ldots 0.00 \ldots \mathbf{0.10} \ldots \mathbf{0.15} \ldots 0.20$
$\dot{\theta}$	$-4.00 \ldots 0.00 \ldots \mathbf{1.00} \ldots \mathbf{2.00} \ldots 4.00$

Table 3: Results of STAQ (Unequal forces)

4.6.2 Longer Pole

The length of the pole in the simulation is increased from 1 m to 1.5 m in this experiment. In a typical run, STAQ reached over 55,000 time periods as the maximum average balancing time of the 20 training instances after 6 splits. The number of iterations for each successive partition is 44, 62, 60, 143, 80, 90, 24, and 112 respectively. The average balancing time of 100 random tests is over 24,000. The controller performs better (in the 100 random tests) than the one with the standard, shorter pole (1 m). This seems to confirm our common-sense knowledge that the longer pole is easier to balance. Table 4 lists a typical partition obtained by the STAQ.

State variables	**Partitions** (New ones inserted by STAQ are shown in bold)
x	$-2.40 \ldots \mathbf{-1.20} \ldots 0.00 \ldots \mathbf{1.20} \ldots 2.40$
$\dot{x}$	$-6.00 \ldots 0.00 \ldots \mathbf{3.00} \ldots 6.00$
θ	$-0.20 \ldots 0.00 \ldots \mathbf{0.10} \ldots 0.20$
$\dot{\theta}$	$-4.00 \ldots 0.00 \ldots \mathbf{1.00} \ldots \mathbf{2.00} \ldots 4.00$

Table 4: Results of STAQ (Variation in pole length)

4.7 Analyses and Discussion

Learning to balance the cart-pole system with automatic quantization is a very difficult task. It is essentially a highly non-linear optimization problem with a large number of variables (e.g., if there are 162 boxes, there are 162 variables with 2^{162} possible control strategies). On one hand, we have demonstrated that the STAQ algorithm can well sustain different dynamic systems by constructing different qualitative state representations, and learn generic control rules based on the newly developed state representations. The more difficult the dynamic systems, the more partitions are developed. On the other hand, STAQ has several weaknesses: the total training time is still too long (hours), and the learned control strategy does not constitute a smooth control surface. We identify possible causes of these weaknesses, and discuss approaches to resolve them in our future research in the rest of this section.

The first reason for overly long training time is due to the nature of the feedback that the system receives. The feedback in the cart-pole system is in the form of binary signal — failure or non-failure. This creates a genuinely difficult credit assignment problem for the reinforcement learning. Since a failure can occur only after a long sequence of individual control actions, it is difficult to determine which actions are responsible for it. A *graded* evaluation of performance which may include the off-center error (or integral error) may provide more internal evaluation signals to reinforcement learning, and thus reduce the learning time and improve the stability of the system (i.e., the cart-pole will be maintained around the center).

The second reason that STAQ takes many trials to learn is that it attempts to learn generic control rules for initial positions in a wide range of the state space. As shown by Sammut and Cribb (1990), the faster the learning program converges to a solution, the more specific the learned control rules are. Voting (Sammut & Cribb, 1990), for example, requires 20 repetitions of BOXES (each may require hundreds of trials) to get a combined, generic, control strategy. Most previous learning approaches (such as BOXES, AHC, CART, ARIC) only learn to balance the cart-pole system from one initial position (except (Anderson, 1986)).

The third reason for long learning time is that STAQ develops both discrete state representation and control rules based upon it simultaneously. If one looks at the average number of trials needed between successive partitions, then STAQ is quite comparable with other approaches (such as BOXES). In addition, some other approaches (such as ARIC and CART) assume previous training or domain knowledge of the dynamic system, while STAQ does not.

As we indicated earlier, the surface of the balancing time in the space of the initial cart-pole position is not smooth, and seems quite chaotic. Some initial positions still have very low balancing time, and a slight change in the initial position can result in drastically different trajectories. One possible cause is that the set training algorithm attempts to improve the average (or sum of) balancing time of initial positions in the sets. If some positions have very long balancing time, others with very short balancing time will not be discovered effectively. One possible remedy to this is to improve the product (instead of sum) of balancing time of the initial positions in the set. The second reason for chaotic behavior of the control strategy is the crystal boundaries between boxes of different actions that result in sudden changes in actions. If the transition of actions in one box to another is continuous, the chaotic behavior could be reduced dramatically. Fuzzy logic (such as the work by Berenji (1992)) can be adopted in STAQ. Finally, the bang-bang control (i.e., non-continuous actions) also contributes to the chaotic behavior. We can interpolate the discrete actions in the surrounding boxes and obtain a continuous action applied to the dynamics system. This approach also results in smooth boundaries between boxes.

Another interesting observation is that the partitions acquired by STAQ are not symmetric. The learning program has no *a priori* knowledge of the symmetry property of the cart-pole system. In fact, symmetric partition is *not* needed, and thus should not be expected, in order to achieve the goal. The controller could first attempt to move the cart to one side of the track and then try to balance the system only within that side. After viewing the real-time graphical display of the simulation, it seems that this is what is happening. However, if some domain knowledge (such as symmetric property of the system) is known in advance, it can be integrated into STAQ.

Some of the early partitions created by the STAQ may not be necessary. Thus, some slices of boxes can be merged to produce more compact control rules. Some previous work has been done (e.g., (Sammut & Cribb, 1990)).

The partitioning strategy of STAQ is quite general; it should work with other reinforcement learning algorithms such as AHC and Q-learning. We have, in fact, made some initial attempts of combining AHC with our partition algorithm. However, we find that AHC sometimes fails to learn to balance the cart-pole system with some other workable partitions. It seems several parameters of the AHC (such as ALPHA, BETA, GAMMA) need to be tuned for different partitions. Further research is under way.

5 Conclusions

Learning qualitative policy from continuous representation to control dynamic systems with unknown models and weak feedback is a difficult task — it is much like an animat, which only receives weak feedback and continuous sensory inputs, attempts to achieve certain goals by developing linguistic terms and policy of achieving the goal based on those terms in a completely unknown environment. We have demonstrated that after many trials, our learning algorithm STAQ develops the qualitative state representation and learns qualitative control rules simultaneously. We have shown that STAQ is able to learn control rules based on different partitions of the state variables for the cart-pole balancing systems with various system parameters. The more difficult the dynamic systems, the more partitions are developed. The learned control strategy also is able to balance the cart-pole for an extended period of time with random initial positions. Several weaknesses of the current method have been identified and possible remedies have been proposed as our future research.

Acknowledgement

The authors thank Malur Narayan and Cecil Lew for help in implementing some of the programs described in the paper. Many discussions with Rich Sutton, Andrew Moore, Jin Jiang, and Zakaria Al Soufi have been very helpful as well.

Reference

Anderson, C. W. (1989). Learning to control an inverted pendulum with neural networks. *IEEE Control Systems Magazine*, *9*(3), 31–37.

Anderson, C. W., & Miller, W. T. (1990). A set of challenging control problems. In Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.), *Neural Networks for Control*, pp. 475–510. MIT Press, Cambridge, MA.

Anderson, C. (1986). *Learning and problem solving with multilayer connectionist systems.* Ph.D. thesis, University of Massachusetts.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, *SMC-13*(5), 834–846.

Berenji, H. (1992). A reinforcement learning-based architecture for fuzzy logic control. *International Journal of Approximate Reasoning*, *6*(2), 267–292.

Berenji, H., Chen, Y., Lee, C., Jang, J., & Murugesan, S. (1990). A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems. In *Proceedings of the 6th Conference on Uncertainty in AI*, pp. 362–369.

Cannon, Jr., R. H. (1967). *Dynamics of Physical Systems*. McGraw-Hill.

Chapman, D., & Kaelbling, L. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of IJCAI-91*, pp. 726–731.

Connell, M. E., & Utgoff, P. E. (1987). Learning to control a dynamic physical system. In *Proceedings of the AAAI-87, Sixth National Conference on Artificial Intelligence*.

Lin, L.-J. (1990). Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of the First International Conference on the Simulation of Adaptive Behavior*.

Ling, C. X., & Marinov, M. (1993). Answering the connectionist challenge: a symbolic model of learning the past tense of English verbs. *Cognition*, *49*(3), 235–290. Anonymous ftp: /pub/ling/papers/cognition-verb.ps.Z on ftp.csd.uwo.ca.

Ling, C. X., & Marinov, M. (1994). A symbolic model of the nonconscious acquisition of information. *Cognitive Science*, *18*(4). In press. Anonymous ftp: /pub/ling/papers/cogsci-nonconscious.ps.Z on ftp.csd.uwo.ca.

Mahadevan, S., & Connell, J. (1991). Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of AAAI-91*, pp. 768–773.

Matheus, C., & Rendell, L. (1989). Constructive induction on decision trees. In *Proceedings of Eleventh International Conference on Artificial Intelligence (IJCAI-89)*, pp. 645–650. Morgan Kaufmann Publishers.

Meyer, J. A., & Guillot, A. (1994). From SAB90 to SAB94: Four years of animat research. In Cliff, D., Husbands, P., Meyer, J. A., & Wilson, S. (Eds.), *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books.

Michie, D., & Chambers, R. (1968). Boxes: An experiment in adaptive control.. In *Machine Intelligence 2 (E. Dale and D. Michie, Eds.)*, pp. 137–152. Oliver and Boyd, Edinburgh.

Moore, A. W. (1994). The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. In Hanson, S. J., Cowan, J. D., &

Giles, C. L. (Eds.), *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81 – 106.

Sammut, C., & Cribb, J. (1990). Is learning rate a good performance criterion for learning. In *Proceedings of the Seventh International Workshop on Machine Learning*. Morgan Kaufmann.

Schlimmer, J. (1987). Learning and representation change. In *Proceedings of IJCAI-87*, pp. 511–515.

Selfridge, O., Sutton, R., & Barto, A. (1985). Training and tracking in robotics. In *Proceedings of IJCAI-85*, pp. 670–672.

Sutton, R. S. (1984). *Temporal Credit Assignment In Reinforcement Learning*. Ph.D. thesis, University of Massachusetts at Amherst. (Also COINS Tech Report 84-02).

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44.

Sutton, R., Barto, A., & Williams, R. (1991). Reinforcement learning is direct adaptive optimal control. In *Proceedings of the 1991 American Control Conference*.

Urbancic, T., & Bratko, I. (1993). Constructing control rules for a dynamic system: probabilistic qualitative models, lookahead and exaggeration. *International Journal of Systems Science*, *24*(6), 1155–1164.

Utgoff, P. (1986). *Machine Learning of Inductive Bias*. Kluwer, Boston, MA.

Watkins, C. (1989). *Learning with delayed rewards*. Ph.D. thesis, Cambridge University.

Widrow, B., & Smith, F. (1964). Pattern recognising control systems. In Tou, J., & Wilcox, R. (Eds.), *Computer and Information Sciences*. Clever Hume Press.

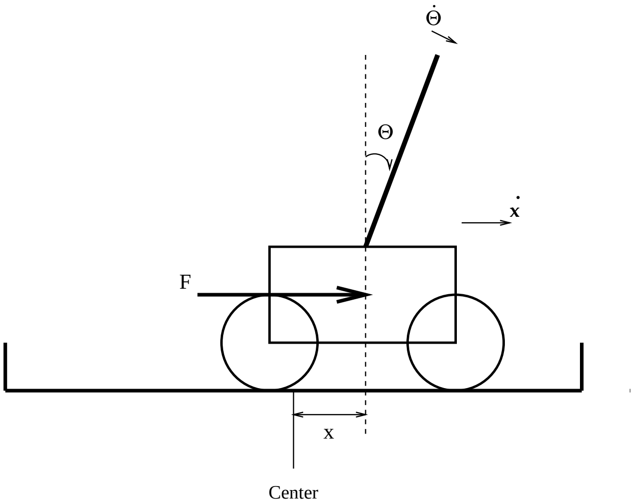

Figure 1: The cart-pole balancing problem

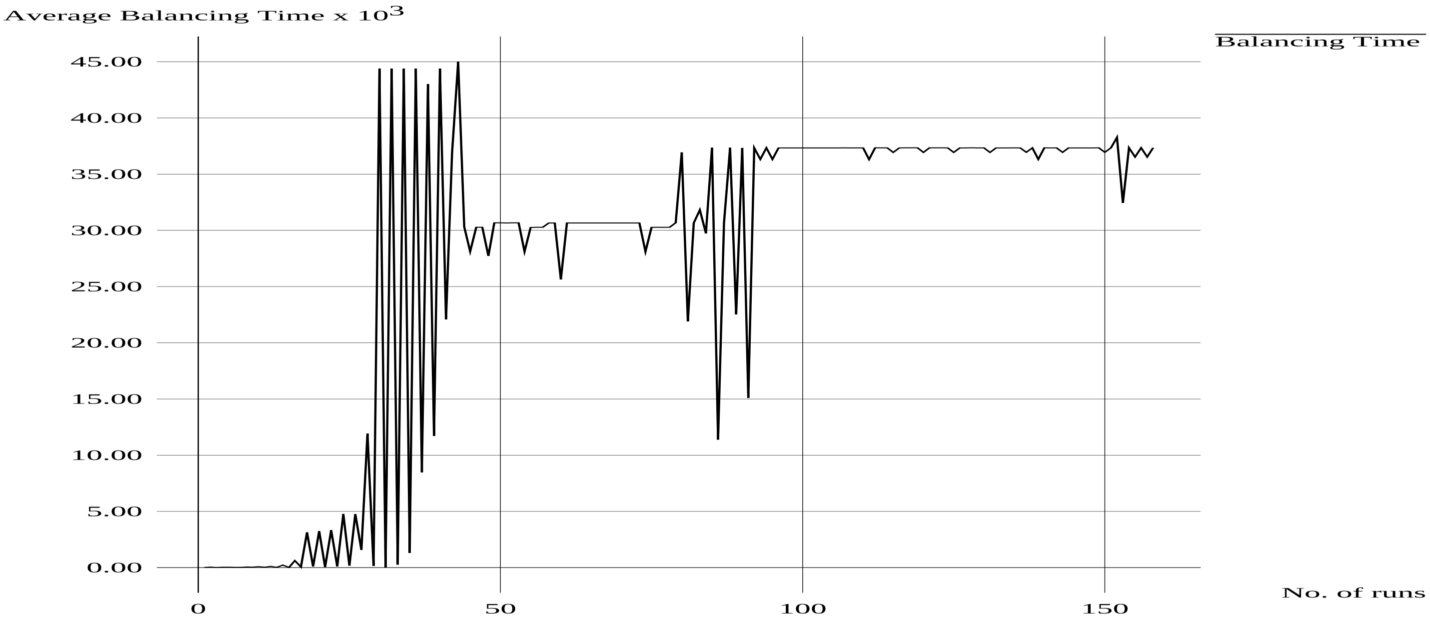

Figure 2: Learning curve of a typical run using the set training algorithm

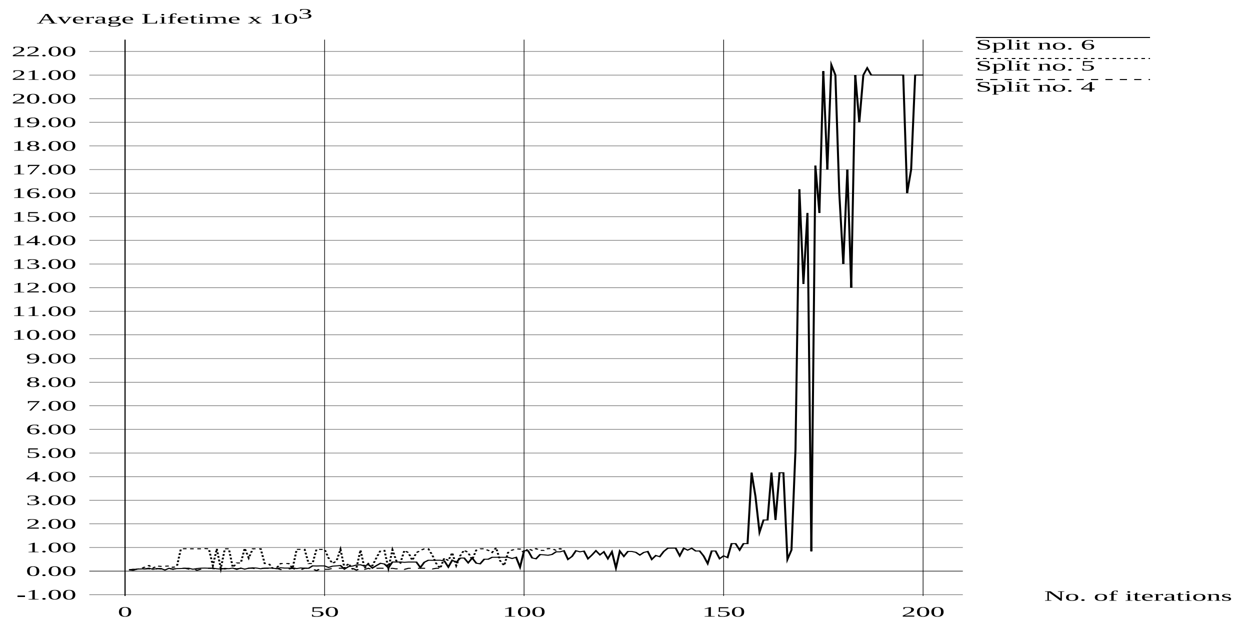

Figure 3: STAQ: Learning curve after 4th, 5th, 6th splits.

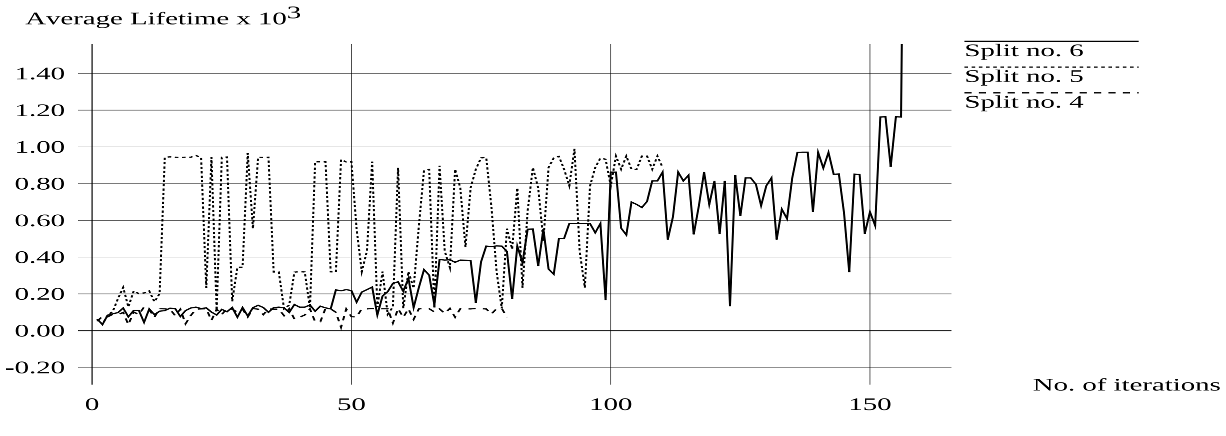

Figure 4: Details of the learning curve after 4th, 5th, 6th splits.