Recent Progress with BOXES

Claude Sammut

School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

1 Introduction

The BOXES algorithm of Michie and Chambers (1968) has proved to be an effective and flexible method for learning to control dynamic systems. The algorithm, in its original form has been used a benchmark for many experiments in control tasks such as pole balancing. Recent work in our laboratory has shown that the BOXES algorithm can be improved to yield very good learning rates. We describe experiments on a variety of update functions and discuss their robustness. We also develop the notion of freezing of BOXES, suggested by Michie and implemented by Bain (1990).

We have also been concerned with synthesising a readable account of the control strategy employed by a set of boxes. Some preliminary work has begun in combining decision tree learning algorithms with BOXES. Using this method, we regard BOXES as the acquirer of sub-cognitive skills and the decision tree induction as a means of introspecting on the learned strategy to generate understandable control rules.

2 Pole Balancing

The BOXES algorithm addresses the problem of learning to control a pole and cart system by trial and error. The physical plant consists of a cart which can run on a track of fixed length. A pole is hinged to the cart such that it can only swing in one dimension. As usually stated, the task only allows bang-bang control. That is, only a force of fixed magnitude can be applied to push the cart to the left or right. The task of the learner is to construct a control strategy that will keep the pole from falling over and the cart from hitting the ends of the track. It is important to note that the problem is to avoid failure rather than to reach a specified target value. A pole and cart system is depicted in Figure 1.

The state of the pole and cart system can be fully determined by the variables: x (the position of the cart), $\dot{x}$ (the velocity of the cart), θ (the angle of the pole), $\dot{\theta}$ (the angular velocity of the pole). That is, the system can be represented by a four dimensional state space.

FIG. 1.1. The Pole and Cart

The most significant problem to be overcome in designing a learning algorithm for this task is that of credit assignment. The control system may make an incorrect choice as to whether to push the cart left or right. However, the consequences of that incorrect choice may not be noticed for some time, when the system finally fails. Many actions may have been taken between the incorrect choice and failure. So how can the learner decide which of those actions was truly incorrect? This is the main task of the BOXES algorithm.

3 BOXES

The algorithm derives its name from the way in which it partitions the state space. The state space is partitioned into regions (boxes) by dividing the range of each dimension into intervals. Thus, the entire space is tessellated by four dimensional boxes. The state of the system determines a box. Each box contains an action setting indicating that when the system enters the box, the action to be performed is given by the setting of the box.

We regard each box as an independent learning element. The task of a box is to learn which action setting is appropriate for that region of the state space. In order to accomplish learning, each box contains statistics on its performance. These are:

- How many times each action has been performed (the action's *usage*).
- The sum of the lengths of time the system has survived after taking a particular action (the action's *lifetime*).

Each sum is weighted by a number less than one which places a discount on earlier experience. This will be represented by, DK, in the algorithms described below.

The BOXES algorithm proceeds by making an initial random selection of setting for the boxes. A trial is performed by using the current box settings to control the system. If the controller fails, then the actions settings in each box are reviewed and possibly changed. These new settings are then used for a new trial. This process repeats until the system can be kept under control for a predetermined time which signifies success. Clearly, the critical operation in the algorithm is the choice of action setting.

3.1 Deciding which action to take

After a failure, a decision is made in each box, whether to set the action of that box to be 'push right' or 'push left'. This decision must trade off exploration versus exploitation. That is, should the setting be chosen such that the most successful action, so far, is chosen or should the learner switch actions if the other action has not been tried very frequently and thus there is little information about its likelihood of success?

To make the choice between actions, BOXES determines the relative merits of pushing left or right and then applies the following rule.

```
if $value_L > value_R$ then
    choose left
if $value_L < value_R$ then
    choose right
if $value_L = value_R$ then
    make random choice
```

The calculation of the value of an action must include the trade-off described above. In the following subsections we will describe a succession of formulas that have been used in this calculation. We begin with the original Michie and Chambers formula and then proceed to describe, in historical order, improvements that have yielded faster learning rates.

3.1.1 *The Michie and Chambers Algorithm*

The original method for calculating an actions merit in a box is as follows. To find the value for 'push left' we use the left action's lifetime (LL) and usage (LU) statistics.

$$value_L = \frac{LL + K \times target}{LL + LU}$$

where

$$target = C_0 + C_1 \times merit$$

and

$$merit = \frac{GL}{GU}$$

K, $C0$ and $C1$ are experimentally determined parameters. GL is the global life time (the sum of the lengths of time the system has survived after taking any action) and GU is the global usage (the number of actions taken). Like the local lifetimes and usages, the global statistics are also subject to decay.

This formula was devised to provide for trade-off between exploration and exploitation. The target was introduced to tie the level of exploration to the overall performance of the system. The learner becomes more conservative in its exploration as the overall performance improves. The Michie and Chambers algorithms was a milestone in learning to control dynamic systems. However, the number of trials required to learn to control the pole and cart is quite high. A number of variations of the original algorithm have lead to improved learning rates. We described these variations and follow these with comparisons of performance.

3.1.2 *Cribb's local merit*

James Cribb (Cribb, 1989) introduced a variation in which each box used a local merit function. Cribb argued that the level of exploration within a box should be tied to the performance of the box rather than the whole system. Thus, he devised the following local merit, replacing the global merit in the Michie and Chambers formula: The local merit is the larger of $\frac{LL}{LU}$ and $\frac{RL}{RU}$. Local merit was found to halve the number of trials to learn to control the pole and cart.

3.1.3 *Variations on local merit*

Cribb's version of BOXES can be simplified further by making the trade-off between exploration and exploitation explicit in the control structure of the algorithm rather than hiding it inside the update formula.

```
if an action has not been tested
    choose that action
else if $\frac{LL}{LU} > \frac{RL}{RU}$
    if $\frac{LU}{RU} < C_0 + C_1 \times \frac{LL}{LU}$
        choose left
    else
        choose right
else
    if $\frac{RU}{LU} < C_0 + C_1 \times \frac{LL}{RU}$
        choose right
    else
        choose left
```

C_0 and C_1 are experimentally determine parameters.

In this variation of the BOXES algorithm, running averages of the lifetimes of actions are compared. Assuming that the left average is greater than the right, the default action is to push left. However, before taking that action, we ensure that ratio of usages, $\frac{LU}{RU}$, does not exceed a target value.

The intuition behind this formula is that the trade-off of exploration and exploitation is related to the ratio of usages of the actions. If the ratio favours the left action this suggests that the left action has been used in preference to the right. If the merit of the box is not sufficiently large, then the program should with the right action.

This variation learns more quickly than Cribbs version as well as being easier to understand.

3.1.4 *Laws Algorithm*

Law (1991) replaced the test $\frac{LU}{RU} < C_0 + C_1 \times \frac{LL}{LU}$ by $\frac{\left(\frac{LL}{LU}\right)}{\left(\frac{RL}{RU}\right)} > \frac{LU}{RU}$.

If the ratio of average lifetimes exceeds the ratio of usages then the action represented by the numerator in the ratios should be favoured. Thus the selection algorithm in BOXES can be simplified to:

```
if an action has not been tested
    choose that action
else if $\frac{LL}{LU^2} > \frac{RL}{RU^2}$
    choose left
else if $\frac{LL}{LU^2} < \frac{RL}{RU^2}$
    choose right
else
    choose an action at random
```

3.1.5 *Variation on Law's algorithm*

The exponent in Laws variation can be seen as an exploration factor. Let us rewrite the algorithm above as:

```
if an action has not been tested
    choose that action
else if $\frac{LL}{LU^k} > \frac{RL}{RU^k}$
    choose left
else if $\frac{LL}{LU^k} < \frac{RL}{RU^k}$
    choose right
else
    choose an action at random
```

As K approaches one, the level of exploration falls to zero. The higher the value of K, the greater the level of exploration. This variation is the most successful version of BOXES to date. The following section describes comparisons between some of the variations.

Table 1.1 Comparison of number trials for variations on BOXES

Law & Sammut	Cribb & Sammut	Michie & Chambers	
79	152	115	
120	80	60	
94	131	140	
69	171	158	
45	90	211	
89	209	747	
80	224	285	
73	105	279	
68	116	2207	
81	166	2586	
81	230	484	
90	138	455	
43	130	387	
43	143	873	
103	85	189	
48	146	505	
81	61	249	
89	112	228	
68	122	392	
65	61	581	
75	134	557	Average
0.29	0.38	0.92	Std. Dev.of log

FIG. 1.2. Determining the value of K

4 Comparison

Three variations of BOXES are compared: Law & Sammut, Cribb & Sammut and the original Michie and Chambers algorithm. Learning experiments were repeated 20 times for each variant. The system is started in random states for each trial. Table 1 shows the number of trials required to learn to control the system for each of the 20 experiments. The average number of trials is shown at the bottom of the table. To study the consistency of the results, logs were calculated. The standard deviations of the logs are also shown at the bottom of the table.

One of the points of comparison is that the Michie & Chambers algorithm can vary considerably in learning rates, whereas the other two variants are more consistent. The Law & Sammut variant has a learning rate comparable to Selfridge, Sutton and Barto (1985).

5 Properties of the Law and Sammut Algorithm

The problem of most learning system for this domain is that their properties are not well understood. In particular, the values of parameters can only be determined experimentally. This section describes some of the properties of the Law & Sammut variant, determined by experiment.

Figure 2 shows a plot of the average number of trials against K values for a series of learning experiments. The average was obtained over five learning runs. The graph shows that the algorithm is stable over the range 1.4 to 1.8. For the standard pole balancing problem, 1.7 was found to be the best.

FIG. 1.3. Determining the value of DK

The DK value in the previous experiments was set at 0.98. While exhaustive experiments have not been completed, Figure 3 shows the results of experiments to determine an appropriate value for DK.

Once workable values for parameters have been found, it is reasonable to ask what are these settings sensitive to. One possibility is that the value of K depends on the number of boxes in the state space partition. Throughout these experiments we have used the standard pole and cart simulation of Anderson (1987). This uses 162 boxes. The original Michie and Chambers set up used 225 boxes. Determination of K was redone using 225 boxes. The results are shown in Figure 4. AS can be seen, the different number of boxes had little effect. While this is not conclusive evidence, it suggests that K does not depend heavily on the number of boxes used in the state space partition.

The final property tested was that of sensitivity to the problem. Often, the parameters in reinforcement learning algorithms are dependent on the learning task. To find out if this was the case with the Law & Sammut algorithm, the problem was varied as follows. Rather than using an equal force to push left and right, asymmetric pushing was used. That is, only half the force was used in a right push as in a left push. This makes the system less easily controlled. Therefore, we expect the learning times to increase. We wish to observe how the learning rate degrades and if parameters must be changed to find the best learning rate. Figure 5 shows the determination of K for the asymmetric pushing task. As can be seen, the best value for K still lies within roughly the same range. However, the best value is 1.4. Thus, K is slightly sensitive to the problem.

Table 2 shows a comparison of the three variants from Table 1 for the asymmetric pushing problem. We also show the Law & Sammut algorithm

Fig. 1.4. Sensitivity to the number of boxes

Fig. 1.5. Sensitivity to the problem

Table 1.2 Comparison on asymmetric pushing

Law (K=1.4)	Law (K=1.7)	Cribb & Sammut	Michie & Chambers	
134	545	43	1382	
562	445	168	487	
120	911	314	1360	
224	123	917	1195	
132	383	2789	3145	
394	101	780	431	
413	1977	253	1768	
83	125	1726	1916	
273	1044	376	709	
821	735	236	816	
249	1155	607	297	
262	611	319	2008	
611	661	214	833	
150	308	265	1643	
278	1179	94	565	
547	227	1125	565	
73	439	157	574	
305	107	97	6493	
669	284	493	5657	
532	517	360	712	
342	594	567	1628	Average

FIG. 1.6. State transitions in BOXES

for $K = 1.4$ and $K = 1.7$. It is interesting to note that, proportionally, the Cribb & Sammut variant does not degrade as much with constant parameter setting as the Law & Sammut variant.

6 Reliable Controllers

Sammut and Cribb (1990) noted that a program that learns to control the pole and cart in a single learning run does not necessarily learn how to control the pole and cart in general. That is, the program has not learned how to control the system, no matter what the starting state is. Instead, it has learned to control the system from one particular start state. Figure 6 illustrates why this may be so.

The dynamic behaviour of BOXES can be characterised by a graph in which the nodes represent boxes and the edges represent transitions from one box to another when a left or right push is performed. Strictly, the edges should have labels corresponding to left and right transitions. It should be noted that the transitions are non-deterministic since the same action in the same box does not guarantee a transition along the same edge. Thus the control of the pole and cart system can be viewed as a Markov process.

The goal of the learning algorithm is to keep the system from entering a fail state, indicated by the heavy circle. In other words, if the learning algorithm can find a closed set (i.e. a set of nodes which the system never leaves, indicated by the heavy arrows) then its task will be accomplished. If the learning algorithm is able to find such a set quickly from a particular starting state, then it may never explore many regions of the state space. Thus, when the system is restarted in a state which has not be explored, the learned control strategy may fail. Indeed, it is very likely to fail. Sammut and Cribb (1990) described a solution to this problem where the system was allowed to learn to control the pole and cart a number of times and

FIG. 1.7. First attempt at incremental freezing

the results of each of these runs were pooled into one control strategy.

6.1 Voting and Incremental Freezing

The results from different learning runs are pooled by a system of voting. Corresponding box in successive learning runs contribute votes for the correct action in that box. A χ^2 test is used to determine when a vote is significant, so when a vote passes the χ^2 test, the action for that box is frozen. Sammut and Cribb report that when 20 to 30 learning runs are collected and then voting is applied, the result is a reliable controller that can control the pole and cart system from any recoverable random starting position without requiring further learning.

Bain (1990) reports on a method called incremental freezing, suggested by Donald Michie, where votes are collected as learning runs are completed and freezing occurs as soon as the χ^2 is passed. We conducted similar experiments with previous versions of the BOXES algorithm which confirmed the efficacy of this method. However, when incremental freezing was attempted with the Law & Sammut variant the results were not encouraging.

Figure 7 shows a plot of the reliability of the controller obtained by incremental freezing. The reliability of the controller is tested by running it 20 times on new pole balancing tasks which all start in random states. If the system successfully controls all 20 runs, then it is deemed reliable. Despite some erratic behaviour, controllers produced beyond about 70 runs are mostly reliable. When this experiment was repeated with a different random number sequence the results were as shown in Figure 8. Thus, the latest version of BOXES causes incremental freezing some problems. It is not yet clear why this is the case.

FIG. 1.8. Second attempt at incremental freezing

7 Discovering Patterns in Boxes

Sammut and Cribb (1990) also reported in progress in making the results of BOXES more understandable. The upper rectangle in Figure 9 shows the settings of all the boxes after learning. A ‘0’ represents a‘push left’ action and a ‘1’ represents a ‘push right’. The set of boxes can be stored in the computer as a four dimensional decision array and it this array that has been reproduced in the figure. The display groups all boxes in the region where the pole is leaning to the far left in the left hand side major column. All boxes in the region where the cart is placed at the far left of the track are grouped in the top major row.

It is very desirable that the output of a learning program should be readable. In that way, humans can learn something as well as the program. Unfortunately, the top display is not very informative. However, Sammut and Cribb noted that there is noise in the display. That is, some boxes have settings that are inconsistent with their neighbours. In quite a number of cases, the setting of a box is not critical and can be either 0 or 1. Therefore, Sammut and Cribb experimented with coercing boxes to conform to a regular pattern. For example, the lower rectangle in Figure 9 shows a cleaned up set of boxes which is reliable. More importantly, the regularity in the boxes permits compression of this representation to the point where the boxes can be read as a simple rule. This was first noted by Makarovic. His rule is shown in below.

```
theta_dot = -inf..-0.87
        push left
theta_dot = -0.87..0.87
        theta = -0.2..-0.017
```

Fig. 1.9. Cleaning up BOXES

```
                push left
        theta = -0.017..0.017
                x_dot = -inf..-0.5
                         push left
                x_dot = -0.5..0.5
                         x = -2.4..0
                                 push left
                         x = -2.4..0
                                 push right
                x_dot = 0.5..inf
                         push right
        theta = 0.017..0.2
                push right
theta_dot = 0.87..inf
        push right
```

7.1 Combining induction and reinforcement learning

In recent experiments, we have tried to improve the method of discovering patterns in boxes by using decision tree induction. The method used is as follows.

- Each box contributes one example to an ID3-like algorithm.
- The description of a box gives the attributes and the left/right decisions are the class values.
- After running the ID3 algorithm, the decision tree is pruned top-down, breadth-first.

- Each non-leaf node in the tree is replaced by a leaf node whose class value is the majority class of the unpruned node.
- If the boxes defined by the decision tree preserve 20/20 performance then pruning of the node is made permanent.
- Otherwise, we try pruning a sub-tree.

20/20 performance refers to the reliability test described earlier. The decision tree that results from this method is shown below.

```
theta_dot = -inf..-0.87
        push left
theta_dot = -0.87..0.87
        theta = -0.2..-0.017
                push left
        theta = -0.017..0
                x = -2.4..-1
                        push left
                x = -1..2.4
                        push right
        theta = 0..0.017
                x = -2.4..-1
                        push left
                x = -1..1
                        x_dot = -inf..-0.5
                                push left
                        x_dot = -0.5..inf
                                push right
                x = 1..2.4
                        push right
        theta = 0.017..0.2
                push right
theta_dot = 0.87..inf
        push right
```

This tree generates a set of boxes which is regular as can be seen in Figure 10. The great advantage of readable control rules is that they can instruct humans about the nature of control. Sammut and Michie (1991) describe the way in which they were able to transpose knowledge of pole balancing to controlling the attitude of an orbiting spacecraft.

8 Continuing Work

Work on improving the readability of BOXES output continues. We have found that the order of pruning the decision tree influences the outcome.

FIG. 1.10. Pattern of decision tree

To avoid this problem, it may be necessary to re-learn the boxes when a pruned decision tree imposes new patterns on the boxes. That is, the boxes specified by the tree are fixed, but all other boxes are subject to further training. It is also important that intervals can be re-defined, that is that adjacent nodes in the tree can be merged. This may be easily achieved by the adoption of more sophisticated decision tree induction programs.

Bibliography

1. Anderson, C. W. (1987). *Strategy Learning with Multilayer Connectionist Representations.* Technical Report No. TR87-509.3. GTE LAboratories Incorporated, Waltham MA.
2. Bain, M. (1990). Machine-learned rule-based control. In M. Grimble, S. McGhee, & P. Mowforth (Eds.), *Knowledge-base Systems in Industrial Control.* Peter Peregrinus.
3. Cribb, J. (1989). *Comparison and Analysis of Algorithms for Reinforcement Learning.* Honours Thesis, Department of Computer Science, University of New South Wales.
4. Law, J. K. C. (1992). *Adaptive Rule-based Control.* Master of Cognitive Science Thesis, School of Computer Science and Engineering, University of New South Wales.
5. Michie, D., & Chambers, R. A. (1968). Boxes: An Experiment in Adaptive Control. In E. Dale. & D. Michie (Eds.), *Machine Intelligence 2.* Edinburgh: Oliver and Boyd.
6. Selfridge, O. G., Sutton, R. S., & Barto, A. G. (1985). Training and Tracking in Robotics. In *Proceedings of the Ninth International Conference on Artificial Intelligence* (pp. 670-672). Los Altos: Morgan Kaufmann.
7. Sammut, C., & Cribb, J. (1990). Is Learning Rate a Good Performance Criterion of Learning? In *Proceedings of the Seventh International Machine Learning Conference*, Austin, Texas: Morgan Kaufmann.
8. Sammut, C., & Michie, D. (1991). Controlling a 'Black-Box' Simulation of a Spacecraft. *AI Magazine*, *12*(1), 56–63.