Planning with Q-Learning using a Classifier System-based Approach

Chen K. Tham*& Richard W. Prager
Department of Engineering,
University of Cambridge,
Cambridge CB2 1PZ,
United Kingdom
e-mail:ckt@eng.cam.ac.uk,rwp@eng.cam.ac.uk

May 23, 1994

Abstract

A classifier system-based (Holland 1986) approach for Q-learning (Watkins 1989), which permits the incorporation of domain knowledge in the form of rules, is proposed. These rules can be viewed as constraints which prevent the agent from taking illegal actions and limit the size of the state-action space, leading to faster learning. The generation and subsequent evaluation of rules by reinforcement learning is an efficient method of rule induction, which, when used with pruning, result in a compact set of rules. We demonstrate how the method can be applied in a blocks world planning task and compare the learning rates of Q-learning and the bucket brigade algorithm.
Keywords: Reinforcement Learning, Classifier Systems.

1 Introduction

Many Artificial Intelligence problem-solving techniques involve *state-space search*. *Planning* can be regarded as a more difficult kind of state-space search where the possible operations and state transitions in each stage of the search are not known at the start, but can only be determined when the state is reached and the preconditions of various actions checked. Each candidate action is then invoked in turn and the process is repeated recursively in the new state until the goal state is reached. Since state-space search and planning are computation and memory intensive when done using symbolic manipulation languages like LISP and PROLOG, they are only feasible for problems with a small branching factor. The solutions found by these search techniques are not guaranteed to be optimal and safeguards have to be built in to prevent looping behaviour where the same state is visited repeatedly. Furthermore, the outcome of each action from each state is assumed to be fixed and known, i.e. the task is not stochastic and a world model is available.

Reinforcement learning techniques based on dynamic programming and temporal difference (TD) methods (Sutton 1984) have been successfully applied for finding optimal

---

*From January 1995: Department of Electrical Engineering, National University of Singapore. e-mail: eletck@leonis.nus.sg

solutions in planning with a world model (Sutton 1990) and game-playing tasks involving search spaces with high branching factors which are either stochastic (Tesauro 1991) or deterministic (Schraudolph, Dayan & Sejnowski 1994). Although some prior knowledge in the form of neural network structure and state space representation is used, these approaches to problem-solving are *tabula rasa*, i.e. the agent learns from scratch by trying various actions at random in order to discover their utilities. While it is not clear that shaping the initial policy of the agent will be useful since the policy is modifiable as the agent performs exploration, unconstrained random actions are unacceptable in real world applications like robot control where certain actions can be damaging. Extensive domain knowledge for solving a particular problem is often available, but is seldom taken into account in reinforcement learning applications. This knowledge can be used to rule out actions which are known to be wrong in order to reduce the size of the search space and achieve faster learning. Domain knowledge can also be used in a world model to predict the outcomes of different actions instead of relying on observations of state transitions to build a world model from scratch, as in Sutton (1990).

In this report, we present a method for incorporating domain knowledge in a Q-learning (Watkins 1989) agent: classifier system-based Q-learning (CS-QL), and demonstrate its usefulness in the classic blocks world planning task. The domain knowledge, given as *prior* knowledge, is encoded as rules in a classifier system (Holland 1986). These rules are used to determine the candidate actions in each state and to predict the state transition that occurs as a result of taking a particular action, i.e. a world model. Instead of using a function approximator, e.g. look-up table or neural network, to store the evaluation function over the state space, a classifier in a classifier system is used to store the evaluation associated with each relevant state-action pair. Each classifier also has a strength value associated with it, indicating its usefulness. We show that by pruning low strength classifiers, a compact and easily-interpretable representation of the solution found by reinforcement learning can be obtained.

2 Reinforcement Learning

Reinforcement learning (Barto, Sutton & Anderson 1983) is an on-line method for solving Markovian sequential decision tasks where the agent operates in a stochastic dynamical environment. A model of the dynamics of the environment is not required and the performance of the agent is judged on the basis of a scalar signal known as the *reinforcement* or *payoff* which it receives from the environment. The objective is to determine the best *policy*, which specifies the agent's actions in each state, in order to maximise some cumulative measure of payoff received over time. An example is the expected *return*, which is the expected long term discounted sum of payoff. The agent performs different actions on the environment in order to discover their utilities and increases the probabilities of selecting promising actions.

Q-Learning

In Q-learning (Watkins 1989), the *state-action value* $Q(x, a)$ is estimated. $Q(x, a)$ is the *return* in state x when action a is performed and the optimal policy is followed thereafter. The action space is discrete and a separate $Q(x, a)$ exists for each action a.

Each time the agent takes an action a from state x at time t, the current state-action value estimate for x and a denoted by $\hat{Q}_t(x,a)$ is updated as follows:

$$\hat{Q}_{t+1}(x,a) = \hat{Q}_t(x,a) + \eta\epsilon_{t+1} \tag{1}$$

$$\epsilon_{t+1} = r_{t+1} + \gamma \max_{l \in A} \hat{Q}_t(y,l) - \hat{Q}_t(x,a) \tag{2}$$

where y is the actual next state, γ is the discount factor, η is a step-size parameter, A is the set of possible actions and r_{t+1} is the payoff the agent receives when action a is taken in state x. The state-action value estimates for other states and actions remain unchanged.

When $\hat{Q}(x,a)$ has converged to the true state-action values $Q(x,a)$, then the *greedy* policy that selects actions according to the following criterion is optimal: $a^*(x) = \arg\max_{l \in A} Q(x,l)$. However, during learning, the agent has to explore by performing different actions. One way is to select actions according to the Boltzmann probability distribution:

$$P(a \mid x) = \frac{e^{\hat{Q}_t(x,a)/T}}{\sum_{l \in A} e^{\hat{Q}_t(x,l)/T}} \tag{3}$$

where T is a parameter that determines the probability of selecting non-greedy actions. It is slowly decreased during learning to make the policy more greedy.

3 Classifier Systems

A classifier system, as described in Holland (1986), has the following components: (1) a list of *classifiers*; (2) an *input interface* with the environment, comprising sensors and feature detectors; (3) an *output interface* with the environment, comprising actuators; and (4) a *message list*, linking the three other components. Each classifier has two main parts: the condition and the action, and other information such as its strength. It represents a rule of the form "if *condition* then *action*" and the strength reflects the usefulness of the classifier to the system for achieving its goal. A classifier becomes active when its condition part is satisfied by a message in the message list, which was either posted by the input interface or a classifier. The active classifier then posts the message in its action part to a new message list which holds messages to be matched in the next time step, causing either an action to be triggered in the output interface or another classifier to be activated.

The organization of classifier systems permits the apportionment of credit in delayed payoff situations, e.g. when a long sequence of rules or actions leads to a payoff only at the end. Credit (or blame) must be assigned to active classifiers in the sequence which have contributed to the final outcome. Holland (1986) proposed the *bucket-brigade* algorithm, where an active classifier has to make a bid and compete for the right to post a message to the message queue. The size of the bid depends on the strength of the classifier. If successful, the classifier gives up a portion of its strength, which is passed back to its predecessors. Classifiers that are active when a payoff is received from the environment get their strengths increased. Using this method, all classifiers in a chain leading to high payoffs become stronger over time. The bucket brigade algorithm is very similar to the TD(λ) (Sutton 1987) family of algorithms. In Section 6, we compare the effect of using each of these algorithms.

4 Classifier System-based Q-Learning

The classifier system was designed to be a flexible rule-based production system, allowing concurrent evaluation of rules which represent competing hypotheses, apportionment of credit to useful rules using the bucket brigade algorithm, and gracefulness in performance despite the introduction of newly-discovered rules among well-practiced ones. The similarities between classifier systems and reinforcement learning have been noted by Sutton (1994) and Wilson (1994). Sutton regards both the classifier system approach and the reinforcement learning approach using the TD algorithm as reinforcement learning *methods* for solving the reinforcement learning *problem*: maximizing long-term payoff in agent-environment interaction. However, we are not aware of any work using classifier systems for reinforcement learning with the Q-learning and TD algorithms which exploit domain knowledge for faster learning.

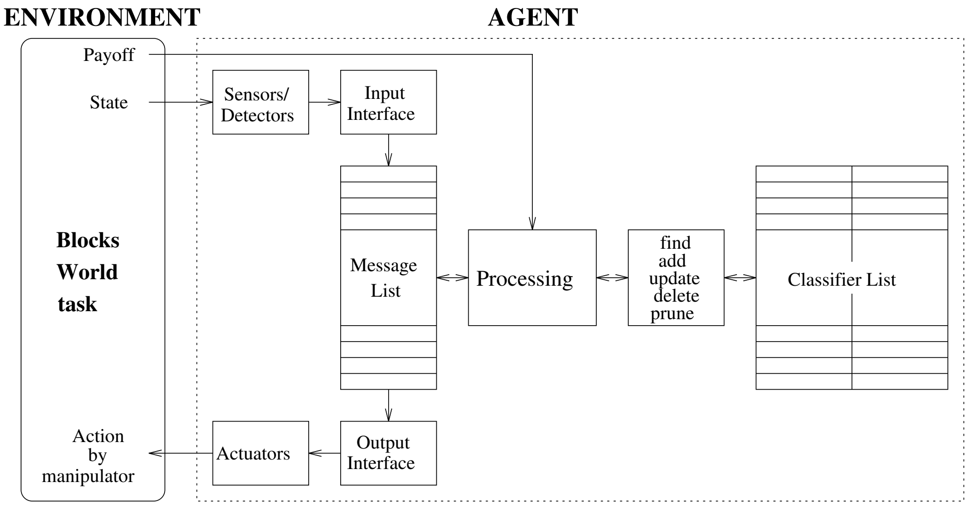

Figure 1: Classifier system for Q-learning.

The classifier system-based Q-learning (CS-QL) architecture is shown in Figure 1. The main difference between this architecture and Holland's original classifier system is the addition of a *processing* block capable of directly manipulating messages and classifiers: this greatly enhances the functionality of the classifier system while maintaining the advantages of a global message list. Instead of matching every classifier with each message, which is an extremely expensive operation, the processing block retrieves particular classifiers depending on the message being processed. The drawbacks of this approach are that the processing block has to be pre-designed and classifiers are not matched directly to messages. One other difference is the absence of a bidding mechanism in CS-QL: all classifiers which have their condition parts satisfied are permitted to have their actions considered, regardless of their strengths. The operations during an agent-environment interaction cycle are shown in Figure 2.

The similarities between classifier systems and Q-learning are also evident at the classifier level. The classifier embodying an "if *condition* then *action*" rule is analogous to a state-action pair with a Q-value $Q(x, a)$. As values can be associated with each classifier, the classifier list can be viewed as a kind of function approximator. The 'don't care' symbol **#**, which matches any value in that bit position, in the condition part of classifiers permits generalization and a more compact representation of rules and function values

1. **Input interface**: Sense the state of the environment and post the state message to the message list.
2. **Process**: Read the state message from the message list and receive payoff from the environment.
3. **Process**: Match the state message with the R-classifiers specifying precondition rules in the classifier list and obtain the set of candidate actions, one for each precondition satisfied.
4. **Process**: Match each state-action pair with the Q-classifiers in the classifier list and verify that a classifier exists for each state-action pair. Create a new Q-classifier for each state-action pair that does not exist in the classifier list.
5. **Process**: Select an action from the set of candidate actions stochastically according to Equation 3. Post the action message to the message list.
6. **Process**: Update the Q-value, prediction error and strength of the classifier which proposed the previous action.
7. **Output interface**: Read the action message from the message list and set the actuators to act on the environment.

Figure 2: An agent-environment interaction cycle: the operations in each block within the agent is described.

since one classifier can cover an entire region in state space. In addition to its strength, we augment each classifier with additional information: Q-value, age, and magnitude of the Q-value prediction error $|\epsilon|$ (see Equation 2) when it was last active. Q-classifiers can be seen in Figure 6. Instead of using a global parameter for controlling exploration, e.g. T in Equation 3, making the degree of exploration dependent on the age and Q-value prediction performance of a classifier covering a particular region of state space can be a useful way of resolving the exploration-exploitation trade-off. However, in the experiments described in Section 6, exploration was controlled only by the gradually decreasing global parameter T.

There are two kinds of classifiers in CS-QL: *R-classifiers* which permit the incorporation of domain knowledge in the form of condition-action rules, and *Q-classifiers* which are state-action pairs used for Q-learning. Unlike fixed-size function approximators like look-up tables where the range and resolution of each dimension and the size of the function approximator have to be predetermined, Q-classifiers are allocated on an 'as needed' basis. In each time-step, if the agent encounters a situation where no Q-classifiers match the current environment state, a new Q-classifier for each possible action is generated for subsequent evaluation. This method of generating new classifiers is different from the *genetic algorithm* approach where classifiers created by genetic operations like crossover and mutation on copies of classifiers with high strength are introduced into the population, displacing weaker classifiers.

In CS-QL, the way in which the strength is determined for each classifier further unifies the two learning paradigms. The strength is given by

$$strength = (1 - |\epsilon|) \tag{4}$$

The value of $|\epsilon|$ is set to the Q-value prediction error when the classifier was last active, but is also incremented by a small amount in each time step. Therefore, only classifiers which predict Q-values accurately and are used often will have high strengths. *Pruning* can then be performed to remove classifiers with strengths below a certain threshold, thus enabling the space used by state-action pairs which are not part of the optimal solution to be reclaimed. A compact and easily-interpretable representation of the solution found by reinforcement learning is also obtained.

5 Blocks world planning task

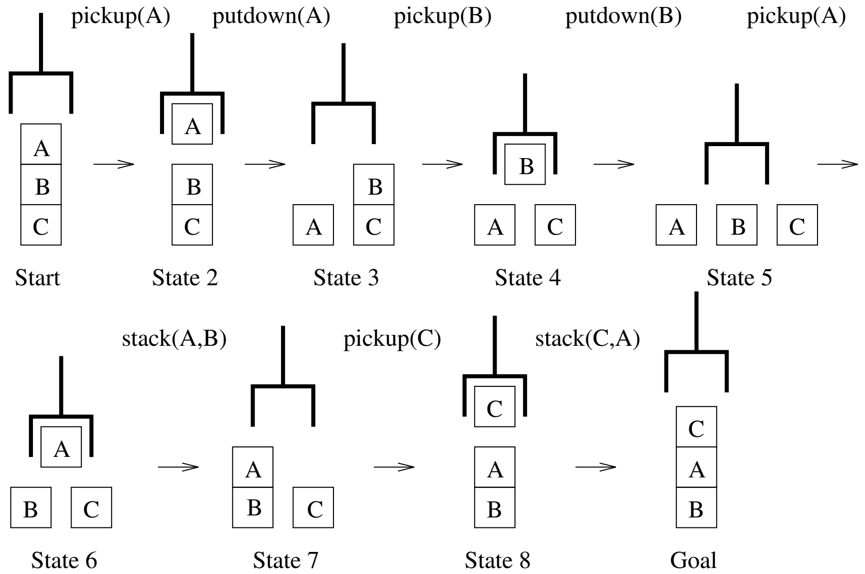

Figure 3: Example of a blocks world planning task with the optimal sequence of actions from a start to a goal configuration. This particular pair of start-goal configurations is referred to as the 'test task' in the text.

The familiar blocks-world planning task is shown in Figure 3. The idea is to plan a sequence of actions to move blocks with a manipulator such that the goal configuration is achieved from a start configuration of the blocks. Planning can be performed using precondition-add-delete lists. Each action has a precondition that is matched against the current state. Only actions that have their preconditions satisfied are treated as candidate actions. In a symbolic AI approach, each candidate action is chosen in turn and the evaluation can be depth-first, breadth-first etc. When an action is applied, the current state is updated to reflect the new circumstances. This is done using the add-delete operations for the selected action: bits in the state message which represent attributes which have become true in the new state are switched on, and those which represent attributes no longer true are switched off. The entire process is repeated recursively in the new state until the goal state is reached.

If the state space consists of all possible configurations of the blocks and the action space all the possible actions, Q-learning can be used to discover the optimal sequence of actions from any start to any goal configuration. The attributes which determine the state of the environment are whether the manipulator is holding a block, whether the top of a block is clear, and how the blocks are stacked, i.e. on another block or on the table. With three blocks, there are 22 possible states and 18 possible actions in total; however, in each state, at most three actions are legal. The meanings of different bits in the condition and action parts of classifiers in the classifier system are shown in Figure 4.

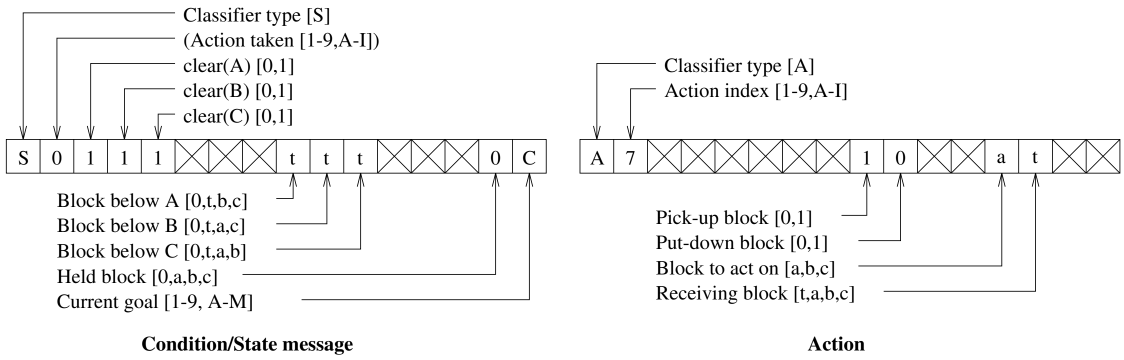

Figure 4: The meanings of bits in the condition and action parts of classifiers, shown for state 5 of the test task and action 'pickup(A) (from the table)' respectively. This format is also used in the state message posted by the input interface and the action message posted to the output interface. ('X' indicates that the bit is not used.)

Preconditions with which state messages are matched in order to determine the set of legal actions are encoded in the 18 R-classifiers listed in Figure 5(a). An additional 18 R-classifiers, Figure 5(b), are used to specify the add-delete operations to update the state message after an action has been taken. These rules form a world model which enables the agent to predict the consequences of its actions[1].

```
T    Condition          Action
R S#1#####b#####0# A10000001000ab00
R S#1#####c#####0# A20000001000ac00
R S##1#####a####0# A30000001000ba00
R S##1#####c####0# A40000001000bc00
R S###1#####a###0# A50000001000ca00
R S###1#####b###0# A60000001000cb00
R S#1#####t#####0# A70000001000at00
R S##1#####t####0# A80000001000bt00
R S###1#####t###0# A90000001000ct00
R S#############a# AA0000000100at00
R S#############b# AB0000000100bt00
R S#############c# AC0000000100ct00
R S##1##########a# AD0000000100ab00
R S###1#########a# AE0000000100ac00
R S#1###########b# AF0000000100ba00
R S###1#########b# AG0000000100bc00
R S#1###########c# AH0000000100ca00
R S##1##########c# AI0000000100cb00
```

(a)

```
T    Condition          Action
R S1############## S001####0#####a#
R S2############## S00#1###0#####a#
R S3############## S010#####0####b#
R S4############## S0#01####0####b#
R S5############## S01#0#####0###c#
R S6############## S0#10#####0###c#
R S7############## S00#####0#####a#
R S8############## S0#0#####0####b#
R S9############## S0##0#####0###c#
R SA############## S01#####t#####0#
R SB############## S0#1#####t####0#
R SC############## S0##1#####t###0#
R SD############## S010####b#####0#
R SE############## S01#0###c#####0#
R SF############## S001#####a####0#
R SG############## S0#10####c####0#
R SH############## S00#1#####a###0#
R SI############## S0#01#####b###0#
```

(b)

Figure 5: (a) Preconditions for each of the 18 actions. (b) Add-delete lists which modify the state message depending on which action was taken.

Let us consider state 5 of the test task in Figure 3. The state is represented in the state message `S0111000ttt0000C` which indicates that the manipulator is not holding a block,

[1]A world model is required for planning, e.g. Sutton (1990). In a purely reactive system, the next state is determined by observing the state of the environment after an action is taken.

the tops of all three blocks are clear and they are all on the table. The preconditions in the list of R-classifiers that this message satisfies are `S#1#####t#####0# A70000001000at00`, `S##1#####t####0# A80000001000bt00` and `S###1#####t###0# A90000001000ct00`, i.e. the legal actions are to pick up either block A, B or C from the table. Instead of choosing each legal action in turn, one action is selected stochastically from the set of legal actions according to Equation 3. When performing planning, add-delete R-classifiers are used to modify the state message depending on the action selected. For example, if 'pickup(A)' is selected, the state message is first modified to reflect this (`S7111000ttt0000C`), and then matched with the add-delete classifiers. The classifier activated in this case is `S7############## S00#####0#####a#` which causes a new state message `S00110000tt000aC` to be posted to the message list, since a `#` in the action part means that the corresponding bit in the state message is used in the outgoing message. This new state message indicates that the top of block A is no longer clear, block A is no longer on the table and the manipulator is holding block A, while attributes pertaining to the other blocks are unchanged. The test task shown requires eight steps in the optimal sequence: pickup(A), putdown(A), pickup(B), putdown(B), pickup(A), stack(A,B), pickup(C), stack(C,A). The solution found by Q-learning for the test task can be seen from the Q-classifiers listed in Figure 6. After Q-classifiers with low strengths have been pruned, there is only one state-action pair left for each state, specifying the action in the optimal policy for that state.

T	Condition	Action	Strength	Q-value	Age	PredErr
Q	S0100000bct0000C	A10000001000ab00	0.998832	0.698069	3.27919	0.001598
Q	S00100000ct000aC	AA0000000100at00	0.999053	0.735004	3.27869	0.001270
Q	S0110000tct0000C	A40000001000bc00	0.999167	0.773756	3.27719	0.001021
Q	S0101000t0t000bC	AB0000000100bt00	0.999216	0.814500	3.27669	0.000805
Q	S0111000ttt0000C	A70000001000at00	0.999215	0.857373	3.26719	0.000601
Q	S00110000tt000aC	AD0000000100ab00	0.999161	0.902499	3.22067	0.000400
Q	S0101000btt0000C	A90000001000ct00	0.999047	0.949999	3.21217	0.000200
Q	S0100000bt0000cC	AH0000000100ca00	0.998859	1.000000	3.19766	0.000000

Figure 6: Q-classifiers for performing the test task.

6 Results

We carried out four sets of experiments with the blocks world planning task to demonstrate the advantages of using a classifier system-based reinforcement learning approach:

1. **LT-QL**: Q-learning with TD(0) is performed and a look-up table is used to store the Q-function. It is necessary to know beforehand that there are 22 states and 18 actions in the problem. In each state, one action out of 18 is selected according to Equation 3. When an illegal action is chosen, the current state remains unchanged; however, in a real world application, this may have serious consequences.
2. **CS-QL**: This uses the modified classifier system explained in Section 4 with Q-learning and TD(0). In each state, Q-classifiers for the legal actions determined in the process described in Section 5 are created if they are not already present. An action is selected from this set according to Equation 3.
3. **CS-BB-BZ**: This is similar to the CS-QL case, but Q-learning is not performed. Instead, Holland (1986)'s bucket brigade algorithm is used to update the strengths of active classifiers. In each state, an action is selected from the set of legal actions

according to Equation 3; however, the strengths of the classifiers proposing the candidate actions are used instead of Q-values.

4. **CS-BB-RW**: This is similar to the CS-BB-BZ case, i.e. the bucket brigade algorithm is used for updating the strengths of active classifiers. Instead of using the Boltzmann distribution of Equation 3, an action is selected by the 'roulette wheel' method commonly used in classifier systems, e.g. Wilson (1994). The probability of selecting an action is equal to the strength of the classifier proposing that action divided by the total strength of classifiers proposing legal actions.

The objectives of the experiments are: (1) find the optimal plan in the blocks world task from any start to any goal configuration, and (2) obtain a compact representation of the solution, as shown for the test task in Figure 3. We performed 50 cycles of 2500 trials for each of the four cases. In each trial, the start and end configurations were chosen randomly. The test task, which involves eight steps in the optimal sequence (longer than the average of 4.5 steps per trial), was run once every 50 trials throughout the 2500 trials to obtain the number of steps required to perform the test task and the average Q-value or strength per step in the test task at various stages of training. These are shown in the graphs in Figure 7. Although there is no generalization and transfer of learning across tasks with different goal configurations, performing the test task once every 50 trials provides a worst-case estimate of the performance of the agent at each stage.

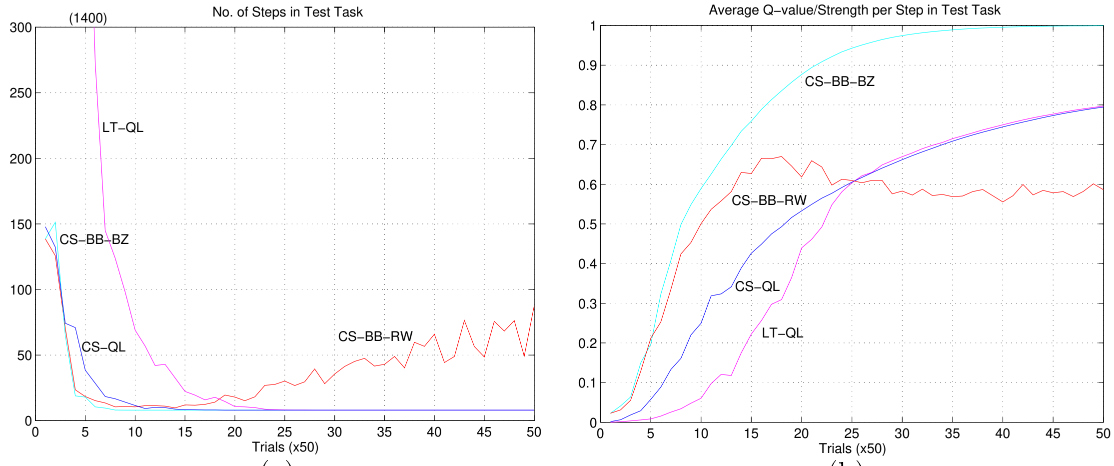

Figure 7: (a) Number of steps required to perform the test task with training. (b) The average Q-value, or strength, per step in the test task with training. In both cases, the result shown is the average over 50 cycles of 2500 trials each. The test task is executed once every 50 trials.

From Figure 7(a), the advantage of having a smaller action space consisting of only legal actions in each state is clear: LT-QL required 1400 steps in the beginning and took 1150 trials to reach the optimal number of eight steps, whereas the other three methods required 150 steps in the beginning and took only 700 trials to reach the optimal number of steps. However, after the 700th trial, the number of steps required by CS-BB-RW increased from eight to about 75 in the test task. This is because *discounting* was not used in the original bucket brigade algorithm and the propagation of payoff from terminating classifiers to classifiers active early in the trial increased the strengths of the latter regardless of whether they were in the optimal solution. Thus, action selection by the ‘roulette wheel’ method became increasingly stochastic with training. This problem did not arise in CS-BB-BZ where action selection was based on the Boltzmann distribution (Equation 3). Decreasing exploratory behaviour during training caused the correct actions which led to payoff in the early stages of training to be selected more frequently than other actions, preventing the latter from being strengthened further.

In Figure 7(b), the average Q-value per step in the test task for CS-QL increased much faster than that of LT-QL until the 625th trial. The curves for CS-BB-BZ and CS-BB-RW increased much faster than LT-QL and CS-QL due to the absence of discounting. However, when the performance of CS-BB-RW began to deteriorate after 700 trials, the average strength per step fell to intermediate levels. After training, pruning with a strength threshold of 0.1 was carried out, resulting in a decrease in the number of Q-classifiers from 882 to 446. When these Q-classifiers are used for planning from any start to any goal configuration, optimal behaviour is always obtained.

7 Related work

Wilson (1994) used a discounted bucket-brigade algorithm and a Q-bucket-brigade algorithm in a zeroth level classifier system (ZCS) to enable an agent to move towards food in a ‘woods’ environment. He noted the strong resemblance of these algorithms to Q-learning. However, the strengths of active classifiers in ZCS were treated as though they were Q-values. With discounting, a classifier which contributed to goal attainment in a ‘setting-up’ role and was active early in the trial had a lower strength than classifiers which were active nearer the goal. This distorts the concept of having the strength of a classifier reflect its usefulness to the system. In CS-QL, the strength and Q-value of a classifier are maintained separately. Under this scheme, ‘setting-up’ classifiers will have the correct combination of high strength and low Q-value.

8 Conclusion

We have presented a classifier system-based approach to Q-learning and demonstrated its usefulness in a planning task. It addresses to some extent the problem of integrating reinforcement learning with other problem-solving and planning methods (Barto 1993). The classifier system architecture allows the incorporation of domain knowledge in a Q-learning agent. The domain knowledge was useful for restricting the action space that had to be searched, leading to faster learning, and as a world model for planning. Although the idea of utilizing domain knowledge can be implemented by other means, using a classifier system allows rules (R-classifiers) and state-action pairs (Q-classifiers) to be stored and retrieved within the *same* data structure. The strength in each classifier was used to prune redundant classifiers so that storage and computation is minimized and a compact representation of the solution found by reinforcement learning is obtained.

Several shortcomings in the original classifier system approach to solving the reinforcement learning problem have also been highlighted. Our experiments have shown that discounting or using an action selection method based on the Boltzmann distribution can be useful. In CS-QL, the precondition and add-delete rules were given as prior knowledge. Although we have not addressed the important issue of employing genetic algorithms for rule discovery, it is conceivable that these algorithms can be used to automatically acquire such beneficial rules.

Acknowledgements

We are grateful to Richard Sutton at GTE Labs. for his comments on classifier systems and the relationship between TD algorithms and the bucket brigade algorithm. Chen K. Tham is supported by an Overseas Graduate Scholarship from the National University of Singapore.

References

Barto, A. (1993), Reinforcement Learning, A series of seminars presented at the Isaac Newton Institute of Mathemathical Sciences, Cambridge, U.K.

Barto, A., Sutton, R. & Anderson, C. (1983), 'Neuronlike elements that can solve difficult learning control problems', *IEEE Transactions on Systems, Man and Cybernetics* **SMC-13**(5), 835–846.

Holland, J. (1986), Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, *in* R. Michalski, J. Carbonell & T. Mitchell, eds, 'Machine Learning: An Artificial Intelligence Approach', Vol. II, Morgan Kaufmann, Los Altos, CA, chapter 20, pp. 593–623.

Schraudolph, N., Dayan, P. & Sejnowski, T. (1994), Temporal difference learning of position evaluation in the game of Go, *in* 'Advances in Neural Information Processing 6', Morgan Kaufmann, San Francisco.

Sutton, R. (1984), Temporal Credit Assignment in Reinforcement Learning, PhD thesis, University of Massachusetts, Amherst, MA.

Sutton, R. (1987), Learning to predict by the methods of temporal differences, Technical Report TR 87–509.1, GTE Laboratories Inc.

Sutton, R. (1990), Integrated architectures for learning, planning and reacting based on approximating dynamic programming, *in* 'Proceedings of the Seventh International Conference on Machine Learning'.

Sutton, R. (1994), Personal communication.

Tesauro, G. (1991), Practical issues in temporal difference learning, Research Report RC 17223 # 76307, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

Watkins, C. (1989), Learning from Delayed Rewards, PhD thesis, University of Cambridge, Cambridge, UK.

Wilson, S. (1994), 'ZCS: A zeroth level classifier system', *Evolutionary Computation*.