An Evaluation of TRACA's Generalisation Performance

Matthew W. Mitchell

School of Computer Science and Software Engineering
Monash University,
PO Box 197, Victoria, 3145,
Email: matt@csse.monash.edu.au

Abstract

TRACA (Temporal Reinforcement learning and Classification Architecture) is a learning system intended for robot-navigation problems (Mitchell 2000). One problem in this area is the input-generalisation problem. Input generalisation requires learning a small set of internal states which represent useful abstractions of the much larger set of actual states. As such, the input-generalisation problem is fundamentally similar to the classical problems of classification, concept learning and discrimination. However, for on-line robot-learning tasks different evaluation criteria is applied than that for batch classification tasks. Specifically, a small number of trials is desirable to reduce the risks of damage to the agent and/or its environment. This may come at the cost of more computation during learning and slightly lower predictive accuracy. Other requirements are the abililty to learn on-line without predefined classes (i.e classes must be learned during training), an efficient adaptable representation and minimal parameter tuning.

This paper describes TRACA's generalisation mechanism in detail and evaluates its performance on a number of common classification tasks. The ability of TRACA to use short-term memory to represent hidden-state is ignored in this comparison as in all the following tasks perceptual aliasing can be overcome by including additional features. On most tasks, TRACA's predictive accuracy is within a few percent of the best performing systems compared and TRACA's result is often achieved with less training experience. The experiments also support claims by Holte (Holte 1993) that a high predictive accuracy (above 90 percent in these experiments) can easily be achieved on many well-known classification tasks which are often used for evaluating learning systems.

1 Introduction

Unless specifically excluded by system designers, environmental inputs generated by robot sensor readings will include features (or attributes) that are irrelevant for a particular task. Consequently, for many tasks with large input spaces it is impossible to represent the domain using enumerative approaches. To cope with such a large state space the agent must generalise - a number of states must be treated as the same or similar (Holland, Holyoak, Nisbett, and Thagard 1986; Chapman and Kaelbling 1991; Kaelbling, Littman, and Moore 1996). For generalisation to be useful, the mapping of inputs to generalisations (internal states) must retain the important features of the state. This is the *input generalisation* problem which requires the correct classification of world states for success (Chapman and Kaelbling 1991).

Input generalisation has been addressed specifically in relation to robot learning problems by CS-QL, the G-Algorithm and U-Tree (Tan 1991; Chapman and Kaelbling 1991; McCallum 1995). These systems attempt to develop generalised representations of their environment. TRACA also addresses the input generalistion problem and in this report is compared to a number of traditional generalisation algorithms.

When comparing algorithms there is a variety of measures that can be used to assess performance (Fahlman 1988; Henery 1994). Given TRACA's intended problem domain the important criteria are predictive accuracy, a small number of training examples (wall clock time) and minimal parameter tuning. Minimal parameter tuning is important if an agent is to be used to solve a variety of tasks or the developer is unfamiliar with the problem to be learned. A small number of training examples is necessary to reduce the risk of damage to the agent.

The next section describes TRACA. Section 3 uses an example to illustrate TRACA's operation and the construction of its network. Experimental results are presented in Section 4 which includes performance comparisons with a number of other learning systems. Finally, an assessment of the results and a discussion of future work is presented in Section 5.

2 The System

TRACA is a reinforcement learning system which uses state information from the environment to determine action selection. This information is passed into the system through the *input interface*, while the decision of the system to take some action in its environment is passed out through the *effector interface*. A further input is the reinforcement learning signal indicating the agent's success or failure at achieving its goal(s) (Sutton 1991).

The current state of the environment is presented on the agent's environmental input interface using bit strings. For each string received, the agent is capable of selecting an action before a new input string is presented indicating the environmental state resulting from the action. In this respect TRACA is similar to Holland style Learning Classifier systems and Drescher's Schema Mechanism which also receive environmental inputs (other than reinforcement) as bit strings (Holland 1975; Drescher 1991).

TRACA's generalisation technique incrementally constructs a network representation from hierarchical structures of components which can in turn be used as components for new (composite) structures. Components are constructed during the discovery of new rules and act to gradually build a representation of useful areas of the search space. The technique is developed to allow acting while learning in stochastic environments which contain irrelevant features (attributes).

TRACA uses a suppression technique which is implemented as message passing through the network. This technique allows the system to represent NOT, using only logical AND combinations of system components reducing the number of possible combina-

3.3.2 for details). Suppression also acts to shift control to nodes in groups higher in the hierarchy.

TRACA's internal model of its environment is based on three types of nodes: *detector*, *effector* and *predictor*. Detector and effector nodes implement the interface between the agent and its environment. Predictor nodes are organised into *groups*, which are generalised internal representations of problem states. Predictor nodes and their containing groups are created as TRACA learns and represent transitions between internal states given the selection of an action. Groups can be combined into AND constructs. Multiple combinations of groups form the hierarchical structures of TRACA's network.

2.1 The Role of Groups and Nodes

Environmental inputs are represented by the *detector* nodes, with one node for each bit position in the input string. Potential actions are represented by *effector* nodes, again using one node to represent each action. The remainder of the network is constructed using the *predictor* nodes created during learning. The purpose of predictor nodes is to maintain information on the relationship between environment states and subsequent states resulting from actions. These subsequent states are the *predictions* of predictor nodes.

Predictor nodes are encapsulated within *groups*. Groups form the vertices of TRACA's network and the links between groups the edges. There are two types of groups: *unary* groups and *join* groups. Unary groups are each connected to one detector node from which they are passed messages indicating the presence of either a 1 or a 0 at that detector's bit position in any received input strings. Join groups are composite groups which are passed input messages from two other groups, each of which can be either a unary group or another join group.

The links between groups connect them only to their immediate superiors and subordinates and allow information passing via a variety of messages. Messages received by groups are typically passed on to the nodes they contain, to other groups they are connected to, or both. One type of message, already mentioned, is that indicating the content of bits in the input string received from the environment. Detector nodes initiate the processing of input strings by passing up messages to the first layer of groups indicating the content in their bit positions. The receiving groups then continue the process by passing on the information to higher layers of groups. This continues until the information has propagated to all groups in the network.

Predictor nodes have links of their own, separate from the links between groups. Each predictor node has two links, a link to a prediction group and a link to an effector. Effectors represent the actions that can be selected by the system. The links to effectors by nodes are used to send the utility estimates nodes maintain as support for particular actions. This support is then used by TRACA for action (effector) selection (see Section 3.4). Unlike the links from groups to other groups, the link between a node and its predicted group is relatively independent of the node's position in any hierarchy. If we view links between groups and their immediate superior and subordinate groups as vertical links up and down a network hierarchy, the links between predictor nodes and their predicted groups can be visualised as horizontal links across the network between hierarchies. These links connect one part of the network to another across time based on input strings received predicted group and the link to an effector are used by a node to update the Estimated Transition Probability (ETP) and utility estimate it holds.

During learning TRACA can select an effector based on the support received by each from nodes. If no effector has support, an action is selected with uniform random probability. To allow utility estimates to be based on actual experience, nodes are given a small number of trials before they send support to an effector or be used in new combinations of nodes.

Groups and predictor nodes each act in two roles. Groups act to represent internal (abstract) states, determined by their subordinates, and as containers for nodes. Nodes store values within groups and also represent relationships between groups based on actions. The two associations nodes have with groups, are the group as a container and the group as a prediction (a node's prediction may or may not be the same group that contains it).

Nodes, combined with their groups, represent probabilistic S-R-S (Situation-Response-Situation) rules. These rules are of the form (s_a, r_x, s_b) where s_a is an initial state, r_x is a response (action) and s_b is the state that results from taking the action r_x while in state s_a. The rules can be interpreted as:

if the state at t is s_a and action r_x is taken
then the state at $t+1$ will be s_b with probability p

Each node corresponds to one of these rules. The antecedent comprises being in state s_a and having the response (action) r_x selected, the consequent is the transition to state s_b. The rules are probabilistic because the consequent may not always follow the antecedent. This could be due to a stochastic environment, or because a rule is a general rule and the antecedent does not capture all the conditions required for reliable prediction. Groups represent the abstract states which correspond to situations as well as aggregating nodes which represent the rules associating actions and resulting situations. The linking of groups through the predictions of nodes allows chains of rules to form.

Because abstract states are generalisations, and a number of generalisations may be applicable at any one time, more than one rule may be applicable at one time. Furthermore, rules may not always be correct, sometimes taking action r_x in situation s_a might not result in the predicted situation s_b. Therefore, nodes store transition probability estimates (ETPs) which indicate the likelihood of the rule being correct given that the conditions of the antecedent are satisfied. Nodes also store value estimates for the rules they represent based on reinforcements received. These value estimates use a recency weighted averaging process to take into account the probability of the rule receiving reinforcements. Both ETP's and utility estimates are updated using sample updates each time the rule is applied. The next section describes TRACA's operation in detail and uses an example to demonstrate the construction of TRACA's network.

3 System Operation

TRACA's operation follows a basic major cycle. The cycle begins when an input string is received. Detector nodes are *matched* when the input string presented has a 1 in their corresponding bit position. Groups receiving input messages from matched detectors are also matched when their input detector is matched. Once all eligible groups are matched, nodes

als can *fire* and send support for their associated effector. The system will then use this support to select an effector. Nodes in matched groups whose effectors were selected can then *execute*, preparing to update their ETP and value estimate based on whether their predictions are matched with the next input string and whether they receive a reward. This match-fire-execute major cycle is presented in Figure 1.

The next section demonstrates this process of rule execution in the major cycle as well as the process of rule creation.

1. **Match** An input string is presented to the system with bits set to 1 matching respective detectors. Nodes are matched when the input detectors for their group are matched.
2. **Fire** matched nodes send support to their associated effectors.
3. **Effector selection** The system selects an effector based on support sent by nodes.
4. **Execute** matched nodes whose effector was selected prepare to update their ETP and value estimate based on the detectors matched in the next match cycle. Groups matched in the next cycle send any returns to nodes which executed in the previous major cycle which predicted them.
5. **Repeat.**

Figure 1: TRACA's Match-Fire-Execute Major Cycle

3.1 A Simple Maze

A trivial maze navigation example is used to demonstrate the building of the network and its operation. In the example, input strings are used to represent the system state as TRACA moves through the maze. This problem is equivalent to a classification task where the classes are initially unknown.

The maze consists of an aperiodic grid with four states (positions) arranged from left to right. The states are identified (from left to right) by the input strings "100", "010", "110", "001" (see Figure 2). The agent is initially placed in a randomly selected state other than the state identified as "001". The agent has two effectors, effector 0 and effector 1. In any state the learning agent can select one of these effectors, selecting effector 0 corresponds to the action Move-East and selecting effector 1 the action Move-West. Selecting Move-East in any state will move the agent one position to the right, selecting Move-West will move the agent one position to the left unless the state is "100", in which case the agent's position will remain unchanged. Each action incurs a negative reward of -0.1 except when the state "001" is reached when positive reward of 1.0 is provided. Receiving the positive reward constitutes the end of the trial. Once the trial is complete the agent is removed from the maze, its internal state is reset (other than estimated transition probabilities and utility estimates) and it is placed in another randomly selected position (other than the reward position) for the next trial. The task is to learn to the optimal path (based on maximising returns) for all initial positions. The system has no knowledge of the meaning of the input strings or the effects of actions prior to learning. It bases its learning on the transitions between states that it experiences and any rewards received. For simplicity the following discussion focuses on the building omitting the use of rewards and calculation of utilities until Section 3.4.

100	010	110	001

Figure 2: The sample maze

3.2 Initial groups created for the sample problem

TRACA initially has one unary group for each detector and zero predictor nodes. The initial position of the agent in this example trial is the left-most state of "100". Network construction starts with this initial input which matches detector DA, and in turn unary group GA, an effector action is chosen at random following which DA and GA are reset to unmatched. In this case, the selection of effector 1 when B is matched moves the system to the state "010" and this input string matches detector node DB and unary group GB. As initially no node exists in the system to predict the matching of GB, given the previous matching of group GA and selection of effector EA, a new predictor node, GA1, is created. GA1 will now predict the matching of group GB the next time detector DA is matched and effector EA is selected.

This first cycle is shown in Figure 3(1). In each of the cycles depicted in Figure 3, matched detectors and selected effectors are highlighted as darker. Matched groups are also shown as expanded to reveal the contained node whose effector was selected and whose prediction is correct. These nodes have lines to their supported effector and prediction.

Continuing the example with the random selection of EA again, the input string "110" is received resulting in the matching of both GA and GB. Since GB has no nodes to predict either GA or GB given the selection of EA, two new nodes are also created in GB. One to predict GA (GB1 predicts this) and one to predict GB (GB2 predicts this). Both these will execute with the next occurance of GB being matched and EA being selected.

Once more EA is selected randomly, and the agent moves East and receives input string "001". Nodes GA1, GB1 and GB2 all execute, and all three nodes' predictions are incorrect, as the groups they predict are not matched. GC is matched, and GA and GB each create a new node (GA2 and GB3) to predict GC next time they are matched and the effector EA is selected. At this point the trial is completed and the agent is removed from the maze.

The groups created so far are insufficient to correctly represent and navigate the maze. If the group GA is now matched and effector EA is selected, one of the nodes GA1 or GA2 will make an incorrect prediction depending on whether the current state is "100" or "110". A similar problem exists for nodes in the group GB. The following section describes how these groups are combined by the creation of a join group which correctly represents the problem domain.

3.3 Join Groups

If learning trials continue, at some stage the input string "110" will be received again and the groups GA and GB will both be matched in the same cycle. If the effector EA is selected while the agent is in this state, the nodes GA2 and GB3 will now correctly predict

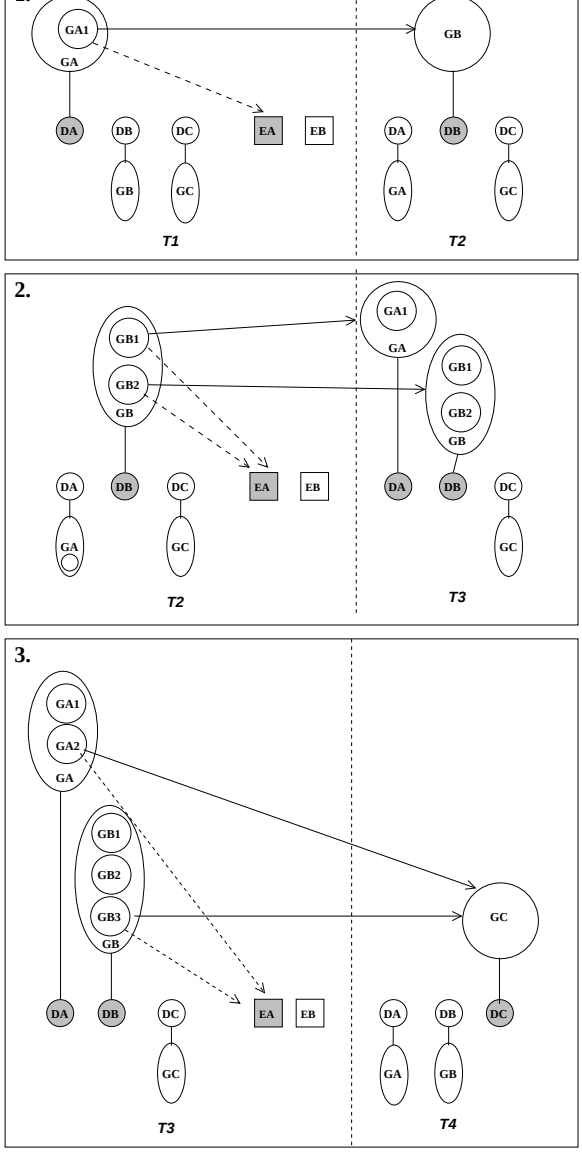


Figure 3: Sequence of predictor nodes forming for maze problem. In each subfigure, shading indicates detectors matched in one cycle (left), the selected effector (center), and the detectors matched with the following input string (right).

the subsequent matching of group GC (all the other nodes in each of these groups which support effector EA will make incorrect predictions).

When a group (such as GC) is correctly predicted by nodes in multiple other groups, it randomly selects and combines two of the predicting groups to create a new join group representing an AND combination. In this case, a new join group, GAB, is created to predict GC given the matching of both GA and GB and the selection of effector EA (Figure 4) [1]. This join group implements a logical AND construct. Its initial node can be interpreted as representing the rule:

if detector DA is matched and detector DB is matched
and effector EA is selected
then detector DC will be matched.

Join nodes construct a hierarchy by combining other nodes. In our example group GAB can be seen

[1] Actually a predicted group will only create joins once unsuspended (see Section 3.2).

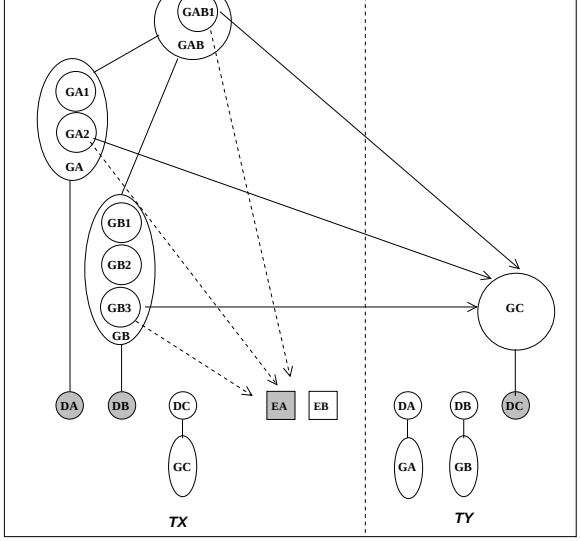


Figure 4: Use of a join group to create an AND construct.

as *superior* to groups GA and GB which are *subordinate* inputs into GAB (Dawkins 1976). GAB will only be matched when GA and GB are both matched in the same cycle.

3.3.1 Node Groups and Chaining

Node groups and nodes implement SRS (situation-response-situation) rules. This type of chaining can occur between groups at any level of the hierarchy allowing representations such as that depicted in Figure 5.

A problem arises with creating SRS rules if negative rewards are being used. When new joins are created, new nodes are also created in other groups to predict them. If one of the nodes in a predicting node's group has a negative value then any nodes created in the predicted group must be given a small number of trials before being used to send returns. This is necessary because a node's value estimate is initially zero and it must converge to a steady-state value. If the true value of the node in the predicted group is a large negative value, but it has not yet converged, it may send returns close to zero. This can disrupt the value estimates of groups containing nodes which predict the group.

3.3.2 Representing NOT

In our example maze, the problem of having nodes GA1 and GB1 incorrectly predict GB and GA respectively when in state "110" is overcome by the group GAB sending a *support suppression* message when it is matched. The support suppression message prevents nodes in the subordinate groups from firing and executing and in our case, prevents the nodes GA1 and GB1 from making an incorrect prediction. In our example, whenever GA1 and GB1 execute without being suppressed they will be correct - their prediction will be matched in the next cycle. The join group in conjunction with the support suppression message now represents a logical NOT, as nodes in each of the groups GA and GB will only fire and execute if the superior group is not matched, and the superior group is only matched when both subordinate groups are matched.

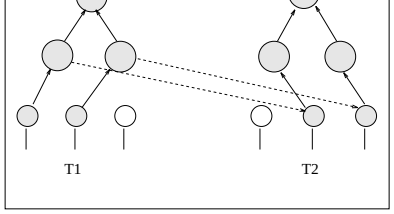


Figure 5: Direct chaining of Group X activated by environmental input (at t_1) to Group Y activated by subsequent input (at t_2).

In general, for the suppression mechanism to suppress a group, GB for example, there must exist another group, GA which is matched when GB is matched and can be used to create a join, GAB of GA and GB. When GA is matched in conjunction with GB the nodes in GAB represent a logical AND of GA and GB. When GA is matched and GB is not matched, the nodes in GA logically represent NOT GB. Note that without GA, it is impossible for TRACA to represent NOT GB. The presence of a group such as GA can be guaranteed in problems where such a group is required by providing one detector position which is matched every cycle (i.e always on).

Using suppression avoids the need for groups which explicitly negate one input which reduces the size and complexity of learned networks. Reducing the number of possible equivalent structures reduces the search space accordingly, in this case by biasing the system from complex structures based on negatives to equivalent simpler structures.

3.4 Calculating Estimates

Omitted from the discussion so far are details of the calculation of the utility estimates and estimated transition probabilities (ETPs). The ETP is updated in each cycle the node executes. The utility estimate is updated for each cycle the node executes and is not support suppressed by a superior group. Because the ETP is updated independently of the suppression messages of superior groups, it is referred to as an *independent* value. The utility estimate is therefore a *dependent* value.

ETPs (e) are calculated using Equation 1. Here r will be 1 if the predicted group had its prediction matched in the cycle after the predicting node's action was selected, and 0 otherwise. The learning rate, α, is a small constant. Initially e is set to zero.

$$e_{k+1} \leftarrow e_k + \alpha\left[r - e_k\right] \tag{1}$$

The learning rule for updating utility estimates is based on Q-learning (Watkins 1989). Environments may have stochastic transitions and/or a stochastic reward function. For Q-learning in environments with stochastic transitions it is appropriate to include transition probabilities in the rule (Peng and Williams 1993). Following is a learning rule for environments with a stochastic reward function (Mitchell 1997):

$$\begin{aligned}\mathcal{Q}_n(x,a) \quad \leftarrow \quad & (1-\alpha_n)\mathcal{Q}_{n-1}(x,a) + \alpha_n \cdot (r + \gamma \\ & \cdot \max_{k \in \mathcal{A}} \mathcal{Q}_{n-1}(y,k) - \mathcal{Q}_{n-1}(x,a)) \end{aligned} \tag{2}$$

where

$$\alpha_n = 1/(1 + visits_n(x,a)) \tag{3}$$

However, the convergence of this rule (due to α_n) prevents adaption in non-stationary environments. This is a problem in TRACA, because general rules will be incorrect sometimes until other rules are developed which suppress it when the specialisations apply. Once the correct specialisation groups are created, the values of nodes in the subordinate general group will rise. From the perspective of this subordinate group, the reward function has changed, so a constant update rate is used in place of α_n. By using a constant rate node utility estimates will reflect changes, but will not converge. Since in practice Q-learning often succeeds with far fewer trials than theoretically required to converge (Littman, Cassandra, and Kaelbling 1995) this should not prevent successful learning and experimental results support this claim. Similar approximations of Q-values are also used by Chrisman (Chrisman 1992) and McCallum (McCallum 1993) for belief states and by McCallum (McCallum 1995) and Ring (Ring 1994) for other internal representations.

However, Q-learning is intended for table-based enumerative state representations. Since TRACA does not use a table representation, but generalised groups, further variations are required. In TRACA's representation, if there are two or more successors occurring simultaneously the preceding group will contain nodes to predict each successor. This situation is depicted in Figure 6(a). Now there is a question as to which successor group's value should be used to update the value estimates of the nodes.

One possible choice is to use the highest value of all the group's successors for an action as the return for each node. Another is to use the value of each node's predicted group to update that node's estimate. The first option requires the transition probability to the successor to be explicitly included in the update rule. The second option - used in TRACA - achieves this same effect in another manner: whenever a successor is not matched its return to predicting nodes is 0. It also returns 0 if it is *support suppressed*.

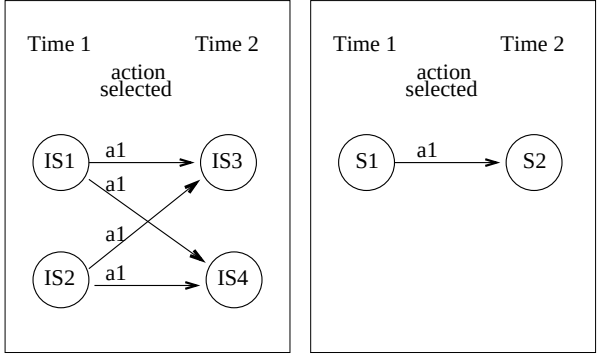
The final update rule for nodes is similar to Equation 2 with a constant update rate and transition probabilities included in a manner similar to that suggested in Peng and Williams (Peng and Williams 1993) and used by McCallum (1995) in U-Tree. However, in TRACA's implementation the recency weighted average return received from the predicted group acts in place of an explicit transition estimate (as in TRACA the transition estimate may also be changing). Whenever a prediction group is matched it returns the maximum value of its nodes. If the predicted group is not matched, or if it is *support suppressed*, it returns zero.

A constant update rate is also used for ETPs. The effect of this is discussed in Section 3.7.

The calculated Q-value is the node's utility estimate and is sent as support for effectors. The ETP is used to determine when join groups should be removed (see Section 3.6).

3.5 Determining the Number of Initial Trials

Nodes are allowed some small number of initial trials in which to learn their value estimates and ETPs. The problem with this is how to determine a sufficient number of initial trials. Both estimates are initially zero and must gradually converge to an asymptotic state. Unless a node receives enough trials to allow its ETP to increase appropriately, subsequent comparisons with its subordinates may cause it to be



(a) Successive states using distributed sensors (a generalised internal model).

(b) Successive states using localised sensors (an enumerative model).

Figure 6: Comparison of internal state models for Q-learning.

removed although given more trials it would demonstrate it is a useful join.

To deal with this problem, the Cox-Stuart test for trend is used to test for the presence of a rising trend in the ETP of nodes. The Cox-Stuart test for trend is a simple statistical test which operates by comparing later observations with earlier observations. If the later observation is higher than the earlier one, this difference is recorded with a plus sign, if it is lower it is recorded with a minus sign. The frequency of these relative signs over a period of time indicates either an upward or downward trend. The probability of the occurrence of different numbers of plus or minus signs allow this test to be used for varying levels of statistical significance (Daniel 1990).

Because the ETPs start at zero, they will initially rise as nodes receive trials. Once the rising trend stops (or if the node's ETP never drops over a minimum sampling period), the node is assumed to have reached its asymptotic state. Once a node reaches this state, its group may be set to unsuspended and it can send support for effectors. Nodes in join groups which reach their asymptotic state can only set their group to unsuspended if:

- the node reached it's asymptotic state without it or any other node in the group having achieved a higher value than their equivalent subordinate (this allows the group to be removed); OR
- the node's ETP has at some point been higher than both its equivalent subordinates and it has had enough trials since it was first higher to determine whether it improves over these subordinates.

The problem that remains now is how to determine if a node improves over it's equivalent subordinates (nodes in subordinate groups with the same action and prediction). If there is no improvement by any nodes in a group the entire group should be removed, otherwise, the group should be retained.

3.6 Removing Groups

Once a group is unsuspended, if it does not provide a improvement over its subordinate groups, it should be removed. The way to determine if a group improves on its subordinates is to compare the ETP's of its nodes with the ETP's of equivalent nodes in the subordinates. If the superior groups ETP's are higher it can be retained, otherwise, it should be removed. However, a simple comparison between the ETP's of superior and subordinate nodes is complicated by the combined effects of a constant update rate and random ordering of the training data.

3.7 Effects of a Constant Update Rate

Because α is constant in both the ETP and the value estimate these values will not converge.

If the value estimate and the ETP's of nodes do not converge they will oscillate around their true value. These oscillations may make a subordinates node's ETP value occasionally rise above that of its superior nodes, even though the ETP value of the superior node is generally higher. This effect will occur when a rule is too general and also in noisy or stochastic environments. The effect is crudely depicted in Figure 7.

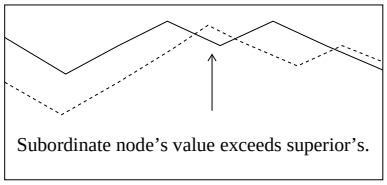


Figure 7: Depiction of oscillating values in a subordinate node and its equivalent superior (same effector and prediction).

3.8 Dealing with Oscillating Values

The test used to tackle the problem of oscillating values is based on the Cox-Stuart test for trend, so it is refered to as the Cox-Stuart based test for noise. What is needed is a method of detecting whether the values in a superior node are in fact consistently lower than an immediate subordinate node, or whether its value is just on a downward oscillation or its subordinate is on an upward oscillation. An average cannot be used for this comparison. The values of nodes change as learned structures develop in the network and start to drive system behaviour, an average

tive is to use a moving average, however, this fails because a relatively small run of events close together can quickly drive values up or down.

The test developed to deal with this is intended to look at events over a period time sufficiently long to offset short trends, yet also reflect longer term changes in values as learning progresses. Since we are dealing with trends, the Cox-Stuart test is used again.

With the test for noise, rather than comparing values at one time against values at a previous time, we compare values of a node in a superior group with the values of its equivalent nodes in the subordinate groups. If the superior node is higher than both its equivalent subordinates (nodes with the same effector and prediction), it is recorded as a plus, if it is lower, it is recorded as a minus. The pluses and minuses indicate the frequency with which the superior has a higher value than its best subordinate. Only if at least one node in a superior group is consistently higher than both its subordinate input nodes (as determined by this Cox-Stuart test) is a group retained in the system. Once retained, only when all its nodes are consistently lower than their equivalent subordinates is the group removed.

The cases for this test are collected as soon as a node has a value higher than both its subordinates and are continued to be collected and evaluated over the lifetime of the node (this allows groups to be removed as described in Section 3.6). This comparison between nodes and their subordinates can be slightly biased by increasing or decreasing the values of either by a small percentage before they are included in the test for trend.

As before, the test for noise is extremely efficient to calculate. For a given sample size and significance level, the detection of a trend reduces to counting the number of pluses or minuses and comparing them to the number required for significance. Alternative measures for significance used in similar learning systems are the student's t-Test (Chapman and Kaelbling 1991) and the Kolmogorov-Smirnov test (McCallum 1995).

3.9 Local Minima

Like any learning algorithm, TRACA is subject to local minima where it finds suboptimal solutions. In TRACA this is most likely to occur because of a *create suppression* mechanism which prevents duplicates.

To address this problem, subordinate groups may be probabilistically excluded from create suppression for individual cycles. This allows other joins to be created using these groups. To avoid duplicates in these cycles, the excluded group's superiors are prevented from being used to create new joins.

3.10 Effector Selection

Two methods of determining the support for effectors are used in the experiments presented in Section 4. These are the *best supporter* and *average support* methods. The *best supporter* method uses as support for each effector the highest utility value of all nodes eligible to send support. The *average support* method uses - for each effector - the average support of all nodes eligible to send support. If both positive and negative rewards are possible in the agent's environment, the *best supporter* method determines an effector support as the highest positive support received minus the lowest negative support received. The *average support* method determines the support the average negative support. In all cases, each group sends as support only the highest positive value and lowest negative value of all nodes in the group.

This completes the description of TRACA's operation. The following section presents experimental results of TRACA on a range of classification and concept learning tasks.

4 Experiments

TRACA is trialled on a number of different tasks. The first three tasks are the Monks concept learning tasks from Thrun *et al* (Thrun et al. 1991). While the Monks tasks are artificial and have only six attributes they do provide a variety of challenges. Following the Monk tasks are four other tasks drawn from real-world problems. The first two tasks involve discriminating different types of irises and voters. The second two tasks, involving larger state spaces, look at TRACA's ability to identify gene splices in DNA sequences and to discriminate malignant and benign cancer cases.

Note, that the same set of parameters is used by TRACA across all tasks, only the number of training epochs is varied. These parameters and the number of learning epochs were not optimised but selected rather arbitrarily based on a small set of initial experiments. In some cases, variations to TRACA's parameters were tried and slightly higher results (up to around 2 percent) were achieved, occasionally these are also reported. In the standard parameter set the Cox-Stuart test used 10 cases and the Cox-Stuart based test for noise used 20 cases with a two-tailed significance of 0.05 and nodes needed to be trialled 20 times before sending returns. The exclusion rate for joins was 1 in each 20 times they were matched (see Section 3.9 for details on this) while the learning rate was 0.1 for all three problems. 1 in 4 actions were selected randomly during training. All results are the average of 20 runs and all, apart from the Monk tasks, are cross-validated. The *average support* method was used for action selection. A summary of the performance on all problems is provided following the experiments.

4.1 The Monks Concept Learning Tasks

The Monk problem is an artificial problem incorporating three different tasks each with its own concept [2]. Each task is a binary classification task involving examples with 6 attributes taking the following values:

- Attribute 1 (A_1): values {1,2,3};
- Attribute 2 (A_2): values {1,2,3};
- Attribute 3 (A_3): values {1,2};
- Attribute 4 (A_4): values {1,2,3};
- Attribute 5 (A_5): values {1,2,3,4} and;
- Attribute 6 (A_6): values {1,2}.

Each example has a value for each attribute and is one state in the total possible space of 432 states. In the artificial domain of this problem each example represents either a friendly or unfriendly robot. For each of the three tasks, membership of the friendly class is determined by the following logical description:

[2] The name is derived from the Corsendonk Priory Monks who invented the problem as a result of hosting the 2nd European Summer School on Machine Learning.

- Monk 2: $A_n = 1$ for exactly two choices of n.
- Monk 3: ($A_5 = 3$ AND $A_4 = 1$) OR ($A_5 \neq 4$ AND $A_2 \neq 3$).

Each task involves presenting a subset of the possible examples to the learning agent along with – or, in TRACA's case, followed by – the information on whether the example is an instance of a friendly or unfriendly robot. From its experience with this subset of the total possible examples, the agent must try to learn how to classify the remaining examples – without being provided explicit information on the class the example belongs to. The third task includes 5% misclassifications (noise) in the training examples.

The Monk 1 and 3 tasks are in a standard disjunctive normal form. However, the combination of attributes for Monk 2 is complicated to describe in disjunctive or conjunctive normal form (Thrun et al. 1991).

Experimental Methodology

The results on the Monk problems of a number of learning systems are presented in Thrun *et al* (Thrun et al. 1991). These systems were all trained on the standard set of training examples for each problem. In each case, the test set was the full set of examples. To allow comparison of results, the same training and test sets are used for TRACA as were used in Thrun *et al* (Thrun et al. 1991). For each problem the numbers of training examples are:

- Monk 1: 124.
- Monk 2: 169.
- Monk 3: 122.

After an example is presented one of two possible system actions is selected indicating TRACA's classification of the example as a member of the class or not. Once the action is selected, another input string is presented with a 0 in all bit positions except one of the last two. Of these two, the leftmost position will contain 1 if TRACA's classification was correct, otherwise, the rightmost position will contain 1. If TRACA was correct a reward of 100 is provided, otherwise the reward is -100.

Since the training sets for all runs on the Monk tasks were identical, each run was started with a different random seed. For Monk 1 TRACA was trained on the training set 30 times (30 epochs) before testing. For Monk 2, 50 training epochs were provided before testing and for Monk 3, 40.

Basic Results

The following briefly summarises performance on each problem before comparing TRACA's performance to that of other learning programs.

On Monk 1(Table 1) TRACA achieved an average of 96.7 percent accuracy on the test data. The average number of groups created and removed during the training epochs was 313.8. This gives some indictation of how many different join combinations were trialled. However, some combinations may have been trialled more than once as TRACA does not prevent the same group being created more than once, it only prevents duplicates existing at the same time. On completion of training there was, on average, 69.3 join groups in the network. Of these groups, an average of 10.9 were unsuspended join groups - groups and which could be used as subordinate components for other joins.

	Test	Removed	Current	Unsuspended
Average	96.7	313.8	69.3	10.9
Std.dev.	3.5	28.1	10.0	4.0
Max	100.0	379.0	93.0	20.0
Min	88.7	262.0	53.0	5.0

Table 1: Performance - Monk 1

For Monk 2 (Table 2), TRACA achieved an average of 70.1 percent on the test data. The average number of join groups created and removed during training was 1091.7, with an average of 117 existing when training completed. Of the existing join groups, an average of 28.8 groups were unsuspended and therefore driving system behaviour.

	Test	Removed	Current	Unsuspended
Average	70.1	1091.7	117.0	28.8
Std. dev.	2.8	46.0	14.3	4.9
Max	76.6	1197.0	149.0	39.0
Min	66.4	1020.0	99.0	22.0

Table 2: Performance - Monk 2

On Monk 3 (Table 3) an average of 92.8 percent accuracy was achieved on the test data. Averages for join groups were 421.6 joins created and removed during learning, with 108.2 in the system on completion of trials, of these 24.9 were unsuspended.

	Test	Removed	Current	Unsuspended
Average	92.8	421.6	108.2	24.9
Std. dev.	3.7	29.7	14.8	5.9
Max	98.2	485.0	127.0	36.0
Min	83.8	377.0	77.0	12.0

Table 3: Performance - Monk 3

Comparing Results to other Learning Methods

The results presented here for comparison are from a number of other learning algorithms selected from Thrun *et al* (Thrun et al. 1991). Other results from Thrun *et al* (Thrun et al. 1991) obtained by non-incremental algorithms have been excluded. The results obtained from incremental algorithms include Back-propagation neural networks with weight decay, Cascade correlation using QuickProp, ID5R, IDR5-hat and IDL. IDL and IDR5 are incremental induction algorithms for constructing decision trees. Two separate results are reported in Thrun *et al* (Thrun et al. 1991) for ID5R. These are included as ID5R*a* for the results obtained by W. Van de Welde and ID5R*b* for the results gained by J. Krueziger, R Hamann, and W. Wenzel (Table 4).

TRACA's predictive accuracy was 96.7 for Monk 1, 70.1 for Monk 2 and 92.8 for Monk 3. The respective standard deviations across the 20 runs were 3.5, 2.8 and 3.7. The three rows in Table 4 after the first contain results obtained by W. Van de Welde. These are averaged over 10 runs for Monk 1 and 5 runs for Monk 2. He did not run experiments on Monk 3 because of the noise. However, Van de

TRACA	96.7	70.1	92.8
IDL	97.2	66.2	
ID5Ra	81.7	61.8	
ID5R-hat	90.3	65.7	
ID5Rb	79.8	69.2	95.3
Backprop. with weight decay	100.0	100.0	97.2
Cascade correlation	100.0	100.0	97.2

Table 4: Comparison of Results for Monk problems

Welde's results are among the most reliable, as many of the other results compared to are all from a single run. In some cases the results were obtained by the algorithm's inventor. For example, the results for Cascade-correlation were obtained by S. Fahlman (Fahlman and Lebiere 1988). His result for Monk 3 is an average of 5 runs.

For the first Monk problem, TRACA's result was up to 15 percent higher than the results for both ID5R*a* and ID5R*b*, and 3.3 percent lower than the two Neural Network approaches. TRACA's results for the second problem were much lower (nearly 30 percent) than the results for both Neural Networks but higher than the results for the other compared algorithms. For the third problem, due to the noise in the data, only three results are available from the compared approaches, Back-propagation, Cascade-correlation and ID5R*b*. Here TRACA's result was 2.5 percent lower than the result for ID5R*b* and 4.4 percent lower than the results for Back-propagation and Cascade Correlation.

Henery (Henery 1994) has criticised the Monk problem for being artificial and Thrun *et al* (Thrun et al. 1991) for including too little analysis of the results presented. Despite this criticism the Monk problem presents more of a challenge to TRACA's classification capabilities than the real-world problems trialled later in this section. W. Van de Welde mentions that the concepts in Monk 2 appear to be too difficult for decision tree based approaches. Interestingly without joins (join creation is turned off) TRACA achieves 92.9 for Monk 3. For Monk 1 and Monk 2 the accuracy without joins is much lower - 72.8 and 67.0 respectively. In fact, of all the problems trialled in this paper, the Monk 1 and Monk 2 problems are the only ones on which TRACA's performance is improved by more than a few percent by using join groups. This is despite TRACA often discovering correct predictive relationships in the data. The following sections examine this issue by looking at TRACA's performance on classification tasks derived from real-world problems and data.

Incidentally, TRACA can obtain an average of 75 percent with a maximum of 81 percent on Monk 2 by varying the parameters. Changes include the addition of a system operating principle to retain groups unconditionally while they have an unsuspended superior, tripling the number of Cox-Stuart cases and doubling the learning epochs. The additional system operating principle appeared to have no adverse effects on the other Monk tasks. In another test, performance on Monk 1 was increased by 1.7 percent to 98.4 by reducing the number of Cox-Stuart cases by half.

4.2 The Iris Discrimination Task

The next task involves discriminating types of Iris based on sepal and petal length and width. There are represented by four attributes in the following ranges (Fisher 1936):

- Attribute 1 (A_1): values {0.2-2.5};
- Attribute 2 (A_2): values {1.0-6.9};
- Attribute 3 (A_3): values {2.0-4.4} and;
- Attribute 4 (A_4): values {4.4-7.7}.

Experimental Methodology

To represent this task for TRACA, the attribute values are coarsely placed into "buckets" of a fixed size for each problem. Ranges covered by bit positions in the respective buckets for each attribute were:

- Attribute 1 (A_1): 0.5;
- Attribute 2 (A_2): 0.1;
- Attribute 3 (A_3): 0.5; and
- Attribute 4 (A_4): 0.1.

For each attribute the first value in the first bucket is the lowest possible value of the attribute, with the last bucket for Attribute 4 ranging from 7.4 to 7.7 inclusive.

Twenty training and test sets were randomly generated from the full data set (150 cases). Each training set contained 2 thirds of the data with the remaining 1 third in the corresponding test set. For each of the twenty runs TRACA was trained for 30 epochs on the training set. As each training set was different and in a random order, the same random seed was used for all runs. Other parameters were the same as in the previous experiments.

Results

TRACA obtained an average of 90.3 on the test sets with a standard deviation of 3.9. For comparison, the inductive learning algorithm C4 obtains 93.8 percent on this data under similar trials (Holte 1993), 3.5 percent higher than TRACA's performance. On average only 5.7 unsuspended join groups (join groups driving system behaviour) were developed during learning. TRACA achieved 88.8 percent under the same conditions but without creating joins.

4.3 The Voter Discrimination Task

Another comparison of TRACA's discrimination and generalisation ability is on a task derived by J. Schlimmer from voting data collected about the votes of Congressmen in the U.S House of Representatives in 1984. This dataset has 435 cases with 16 attributes. Each attribute has three possible values (yes, no or unsure). Each case belongs to one of two classes, either democrat or republican.

Experimental Methodology

Twenty training and test sets were randomly generated each with two thirds of the cases in the training set and one third in the test set. As different training and test sets were used for each run, the same random seed was used. Parameters were unchanged from the previous experiment.

TRACA achieves 92.7 percent on this task with a standard deviation of 2.3. An average of 53.7 unsuspended groups were created. Holte (Holte 1993) reports C4 achieving 95.6 on this task – 3.1 percent higher than TRACA – which was the highest performance from a number of algorithms compared in that paper. When the same experiment was performed with the ability to create joins groups turned off, TRACA achieved an average test accuracy of 94.1, higher than its result using joins and only 1.5 percent lower than C4.

Explanation of Results

An accuracy of 95.6 percent can be achieved on the votes dataset by simply testing the value in attribute 4, *physician-fee-freeze*. This is the most accurate decision tree possible for this dataset (Holte 1993; Buntine and Niblett 1992).

Since TRACA develops rules for each attribute, a system which has rules based on individual attributes (i.e without joins) should be capable of achieving 95.6 percent accuracy. However, TRACA has rules not only for attribute 4, but all attributes as a minimum rule set. A higher result without joins than with joins may occur if joins are incorrectly retained due to ordering effects in the data or if relationships discovered for the training set do not apply to the test set.

4.4 The DNA Splice Task

This dataset includes 3186 examples each representing a window of 60 nucleotides drawn from a processed version of the Irvine Primate splice-junction database. Each nucleotide is represented by one of the symbolic values (a, c, g, t). The task is to identify whether each example represents a intron-extron boundary, an extron-intron boundary or neither. These boundaries occur in DNA sequences where superfluous DNA is removed during protein creation, the extrons are retained in this process and the introns are removed. The processing applied was the conversion of the 60 attributes into 180 binary attributes, removal of 4 spurious examples and the conversion of the symbolic class labels to numeric labels (Michie, Spiegelhalter, and Taylor 1994).

Experimental Methodology

The 20 training sets each had 2000 examples while the test sets each had 1186 examples. In each run TRACA was allowed one pass (one epoch) on the training data before being tested. Other parameters are identical to those used for the Monk, Iris and Voter tasks in Sections 4.1, 4.2 and 4.3.

Results

TRACA achieved an average of 94.9 percent classification accuracy on the test data with a standard deviation of 4.2. On average of 105.6 unsuspended join groups were created. In trials without join groups TRACA achieved an even higher result of 96.8.

In results presented in Michie *et al* (Michie, Spiegelhalter, and Taylor 1994) the best performance was for Radial Basis Functions which achieved 95.9 percent accuracy. Backpropagation Neural Networks achieved 91.2 percent accuracy and C4.5 achieved 92.4 percent accuracy.

The next task uses data from the University of Wisconsin Hospitals collected by Dr. William H. Wolberg. The database consists of 9 attributes for cases of malignant and benign breast cancers. Each of the attributes has a value between 1 and 10. The database has increased in size as more data has become available. The version used in these experiments has 699 cases of which 458 are benign and 241 are malignant.

Experimental Methodology

The parameter settings were as in the previous experiments but with only one training epoch. The 20 training sets each had 400 examples (57 percent of the data) while the 20 test sets each had 299 examples. In each run TRACA was allowed three epochs on the training data before being tested.

4.6 Results

TRACA achieved an average of 93.7 percent classification accuracy on the test data with a standard deviation of 2.0. On average 5.3 unsuspended join groups were created. Without using join groups TRACA achieved a result of 91.4.

The comparison results were obtained by Bennett and Mangasarian (Bennett and Mangasarian 1992) using an earlier version of the database with 566 cases. That database had approximately the same ratio of malignant and benign cases (within 3 percent). The average accuracy over 10 runs of their linear programming discrimination algorithm was 97.4 using 67 percent of the data for training.

With only 1 in 10 actions selected randomly during learning, TRACA achieved an accuracy of 96.0 percent, 1.4 percent lower than Bennett and Mangasarian (Bennett and Mangasarian 1992). Without using joins an accuracy of 91.4 was achieved.

4.7 Effects of Parameter Changes

In the experiments above the same parameters were used apart from the number of training examples (epochs). Enough examples must be experienced for the correct structures to develop and be tested.

Of the other parameters, the learning rate is one of the most interesting, as increasing this may allow nodes to approach their asymptotic values sooner and therefore complete testing sooner. However, having it too high may cause wild fluctuations in values. The effect of varying the learning rate is examined in Section 4.9.

Another interesting parameter is the exploration rate. Determining the rate of exploration required for efficient learning is quite difficult, there are also questions about when exploration should reduce or be stopped entirely. The next section analyses some of the effects of varying the exploration rate. As the Monk 1 problem is one task where joins are necessary to improve performance and on which TRACA performed well, that task is used exclusively for the following experiments.

4.8 Effects of Varying the Exploration Rate

The exploration rate has two major effects on TRACA's learning. One is to influence the allocation of trials to joins so they can be evaluated quickly, the other is to overcome possible overfitting. Both these situations are described in this section.

during learning allows rules to be tested more frequently independently of structures currently dominating system control and behaviour. However, too much exploration can also allow some nodes to appear more useful than they really are. This occurs because nodes higher in the hierarchy execute less frequently than nodes lower in the hierarchy. The greater specificity of the higher nodes protects them from executing incorrectly as frequently as the lower nodes. This specificity has less effect when the diversity of experience is reduced.

Higher exploration rates are also likely to increase the learning time. Learning times increase when nodes in useful joins are not frequently executed. Before a join can be evaluated to be retained or removed from the system, sufficient statistics must be collected for nodes within the join group. For this to occur the nodes must execute a number of times, (i.e they must be matched, have their effectors selected and correct predictions). When exploration is low, nodes better representing correct concepts may be able to drive system behaviour such that nodes in joins based on them execute more frequently.

Another problem occurs if the system does not evaluate joins frequently. A large number of joins form for many different combinations of attributes. Many of these joins are of little usefulness, however, determining this takes more trials when the system's behaviour is not being driven by successful joins. Again, this occurs because a wider variety of actions and predictions are being experienced. Having a larger number of joins which remain in the system for a longer time reduces the opportunities for more useful joins to be created as subordinates are unavailable for new joins to form (they are suppressed during the evaluation of their existing superiors). This problem is one difficulty TRACA encounters on the Monk 2 problem.

Both these effects appear in results for varying the exploration rate on the Monk 1 problem. Table 5 shows the average classification accuracy on the test set (*Test Accuracy*) and the average number of unsuspended join groups developed (*Unsuspended*) as the rate of selecting random actions per classification is changed. A rate of 1.0 indicates that every action was randomly selected, a rate of 0.1 indicates that 1 in 10 actions was randomly selected. The very high exploration rate of 1.0 leads to many structures being retained as more useful structures cannot drive system behaviour to demonstrate their usefulness. Similarly, a very low exploration rate of 0.0 also has slightly more rules as poorer rules are not tested independently of the controlling rule set and they are retained in the system.

Exploration Rate	1.00	0.25	0.10	0.00
Test Accuracy	85.8	96.7	96.2	96.2
Std. dev.	4.0	3.5	5.6	3.7
Unsuspended	22.0	10.9	9.4	14.1
Current	97.0	69.3	64.3	75.1
Removed	271.0	313.8	325.9	289.9
Max	94.9	100.0	100.0	100.0
Min	77.8	88.7	80.8	85.9

Table 5: Performance on Monk 1 when varying exploration rate (averages of 20 runs).

As the learning rate is increased value and transition estimates oscillate more around their asymptotic value. The results of varying the learning rate for Monk 1 are presented in Table 6. Overall, it appears that TRACA's learning of generalisations is robust to changes in the learning rate. As the learning rate increases above 0.1, predictive accuracy falls slightly and standard deviations increase. The number of joins created and removed during learning increases while the number in the system at any one time decreases. This suggests that many joins are being retained for longer periods with lower learning rates than with higher learning rates. The number of unsuspended joins also falls slightly as the learning rate increases. An investigation into each of the runs revealed most of the unsuspended joins were those required to represent the true concepts and that these were retained once discovered. This might be expected for joins whose ETP's asymptote is 1.0, but is more difficult for other joins which represent the true concepts.

Learning Rate	0.05	0.1	0.3	0.5	0.6
Test Accuracy	95.5	96.7	95.4	95.4	94.2
Std. dev.	3.5	3.5	5.0	6.0	4.7
Unsuspended	11.4	10.9	8.1	7.0	6.5
Current	71.7	69.3	57.7	54.1	53.4
Removed	281.2	313.8	399.6	427.7	430.9
Max	100.0	100.0	100.0	100.0	99.8
Min	87.7	88.7	81.9	82.6	83.8

Table 6: Performance on Monk 1 when varying the learning rate (averages of 20 runs).

4.10 Effects of Increasing the Number of Cox-Stuart Cases

The number of Cox-Stuart cases was varied for the Cox-Stuart based test for noise (in the following the number of cases for the Cox-Stuart test for trend is half the number for noise). While these are statistical tests, in reality a higher number of cases should make it easier for joins to be retained in the system as they are less susceptable to being removed due to atypical short runs of events. This claim appears to be supported by the experimental results presented in Table 7. The number of cases is reduced from 40 down to 4. As the case count decreases the number of join groups retained in the system also decreases slightly. For Monk 1 the predictive accuracy peaks at 10 cases, dramatically decreasing when only 4 cases are used. The predictive accuracy of 98.4 for 10 cases reflects the fact that optimisation of parameters was not performed for each of the initial experiments in Sections 4.1 to 4.4. Rather, a generic set of parameters that was anticipated to provide reasonable performance across a range of problems was selected and used.

4.11 Changing the Action Selection Strategy

In the experiments so far, the action selection strategy has been *average support* where all unsuspended, un-support suppressed nodes have sent support and the effector with the highest average support would have its action executed. The alternative method of *best supporter* selects an effector which receives the highest support from a single node. The average support method may better reflect uncertainty when ac-

Test Accuracy	92.8	96.7	98.4	71.8
Std. dev.	5.5	3.5	2.1	2.1
Unsuspended	11.2	10.9	10.8	1.1
Current	71.3	69.3	64.5	36.6
Removed	240.9	313.8	399.9	478.1
Max	100.0	100.0	100.0	75
Min	83.1	88.7	91.7	68.1

Table 7: Performance on Monk 1 when varying the number of Cox-Stuart cases (averages of 20 runs).

curate specialisations of general rules are not possible or have not yet been developed. A comparison of the predictive accuracy using both methods is presented in Figure 8. From this comparison it appears that there is very little difference in performance on the Monk problems for the two different methods.

	Monk 1	Monk 2	Monk 3
Best supporter	93.3	70.4	93.2
Average support	96.7	70.1	92.8

Table 8: Test accuracy on Monk problems when changing the action selection strategy (averages of 20 runs.)

4.12 Making Utile rather than Perceptual Comparisons

The results presented so far have all been based on TRACA retaining groups which demonstrate their ability to predict percepts (better ETPs). However, it is also possible for TRACA to retain groups based on their ability to predict utility or reward. Making this switch would change TRACA from a system which makes comparisons based on perceptual improvements into one which makes comparisons based on utile improvements (McCallum 1995). In the simplest case, this involves replacing the ETP values used nodes for the Cox-Stuart test for noise (described in 3.8) with the absolute utility estimate of nodes. The results of a version of the program with these changes is presented in Table 9. These results were produced under similar experimental conditions as the Monk results presented in Section 4.1 but based on utile improvements.

	Monk 1	Monk 2	Monk 3
Average	94.9	66.6	94.5
Std. dev.	5.2	2.4	2.9
Max	100.0	70.1	98.2
Min	86.1	60.9	88.2

Table 9: Performance on Monk problems making utile distinctions.

The utile improvement results for Monk 1 are very similar to the perceptual improvement results with the utile result only 1.8 percent lower. The utile result for Monk 2 was 3.5 percent lower than the perceptual result, while the utile result for Monk 3 was 1.7 percent higher than its perceptual result. The standard deviations for the two sets of experiments were within 2 percent of each other.

TRACA's predictive performance has been compared over seven tasks using a single set of parameters. Apart from Monk 2, TRACA's predictive accuracy is no more than 4.9 percent below the best performance, and is often well above the performance of a number of other learning algorithms.

As well as predictive accuracy, TRACA's criteria for success includes the requirement for a small number of training examples. While little attempt was made in these experiments to determine the minimum number of required training examples, it appears that the higher predictive accuracy of Neural Networks is achieved at the cost of many more training examples. For the Monk 3 task, Cascade Correlation using Quickprop required 259 epochs, while TRACA's result was achieved within 40 epochs. In one task, TRACA was provided only one pass of the training set.

The small number of joins (relative to the large number of possible combinations) created by TRACA even for large input spaces is also encouraging. The performance of TRACA without joins and with small numbers of joins is consistent with the analysis by Holte (Holte 1993), who determined that many common machine learning tasks require very few rules to achieve good performance. This explains the high accuracy (usually well above 90 percent) on many data sets without using joins. The tasks which most required internal structure for good performance appeared to be the artificial Monk tasks. Whether or not real-world tasks are commonly so simple appears to be an unanswered question.

The relative simplicity of TRACA and minimal intervention of the experimenter is important for the intended application of TRACA to – as much as possible – learn autonomously and be able to solve a diversity of problems. TRACA avoids the common requirement of Neural networks for a high level of expertise by the experimenter (Michie, Spiegelhalter, and Taylor 1994). The fixed set of parameters consistently achieved over 90 percent on a wide range of problems. This ease of use is aided by the fact that the size and topology of TRACA's solution network is determined entirely during learning. While Cascade Correlation also allows for the automatic addition of new nodes as required, due to its batch training requirement, it is unsuitable for the on-line learning environments TRACA is intended for (Fahlman and Lebiere 1988; Lin 1993).

TRACA's quick training times are most likely due to the relative independence of nodes within TRACA's network. It is intended to do more experiments to determine to what extent this allows TRACA to avoid problems inherent in Back-Propagation neural networks such as interference (which results in long learning times) and catastrophic forgetting (Fahlman 1988; McCloskey and Cohen 1989; French 1999). It is hoped that avoiding these problems will allow a single instance of TRACA to learn multiple tasks and re-use knowledge gained on one task for other similar tasks.

Appendix A - Details of the Generalisation Algorithm

Detailed steps of TRACA's generalisation are presented below, along with a class diagram (Figure 8) and sequence diagram (Figure 9). Associations on the class diagram indicate the primary direction of messages between instances of classes. In the case of nodes the association roles are shown to distinguish

and nodes and their predicted group. Association roles are also shown for groups to indicate that the `sendsMessagesTo` association is directed at subordinate groups . The `sendMessagesTo` association encapsulates the two types of suppression messages (the *create suppress* and the *support suppress* messages).

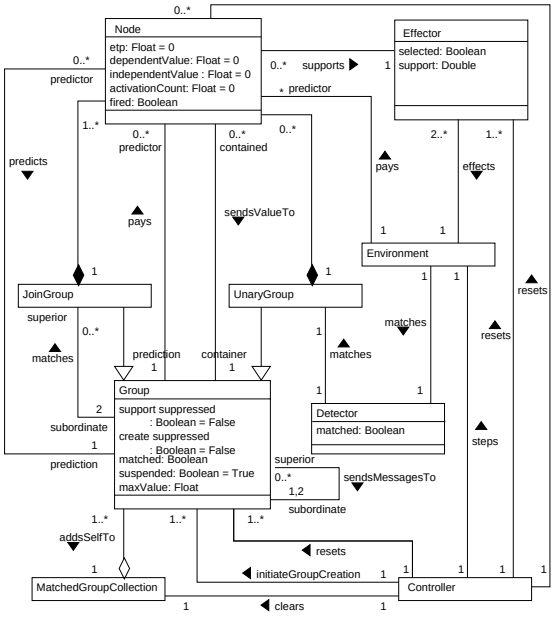


Figure 8: Class diagram showing class relationships for generalisation

Detector nodes receive their inputs from the environment when the major cycle starts as instructed by the system `controller`. Each major cycle is illustrated as a single step in the environment. Detectors in turn send input messages on to the unary groups, which pass those messages up to their superior join groups. Nodes send their values to their containing group so the group can send the highest value to their predicting nodes when they are correct. Correct nodes also receive any instantaneous reinforcement achieved in the environment as a result of their actions. While the controller maintains a list of all nodes and groups, various other lists may be kept for efficiency to keep track of nodes and groups in different states. One of these lists is the list of groups matched in the previous cycle.

The sequence diagram shows the interactions between objects of the various classes. Due to the complexity of depicting multiple instances of groups and nodes in one diagram, a single life-line is used to represent multiple group and node instances. In this sense the semantics of the life-line in the sequence diagram differs from the standard, and is closer to the semantics of a class in a class diagram. Following the sequence diagram is a textual description of significant steps within the major cycle. Some of these steps use backup values from the previous cycle.

Generalisation Algorithm:

The system starts with a controller, a set of effectors, a set of detectors, and a set of unary groups. There is one unary group for each detector. Unary groups are superior to their detectors and initially contain no nodes. Groups are initially suspended until they have had trials to develop useful values

The `support suppressed` *variable is used to both prevent nodes sending support for effectors*

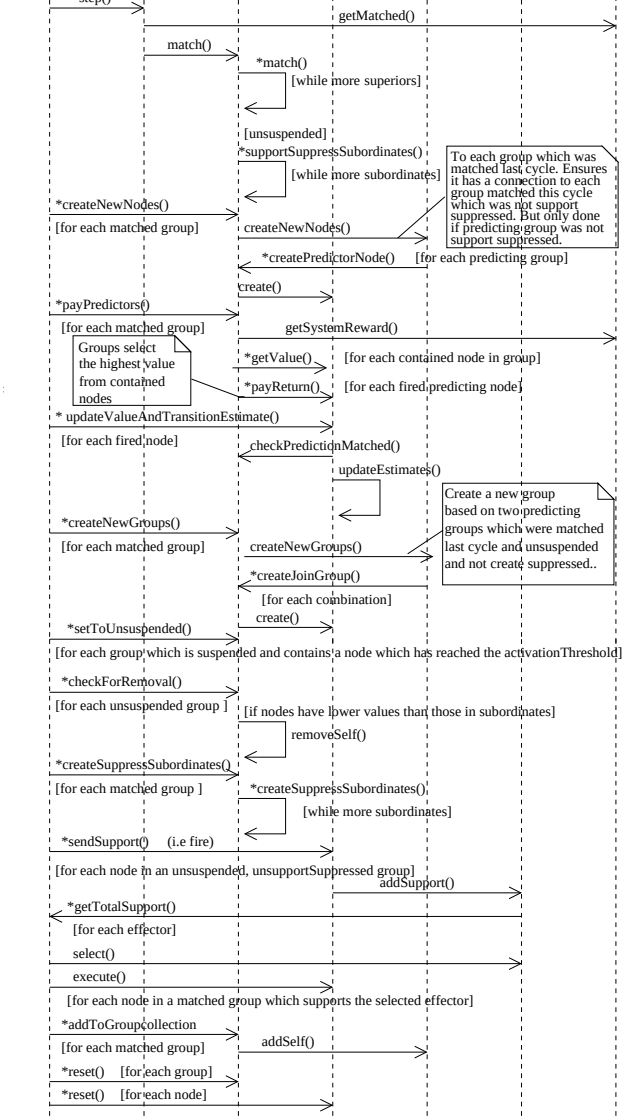


Figure 9: Sequence diagram for the generalisation major cycle

and to prevent groups connecting or being connected to other groups. The `create suppressed` *variable is used to prevent duplicate joins by preventing groups being used as subordinates for new joins. The first is set only by suspended superiors, while the other is set by any superior.*

1. An input string is received by the system. Detectors determine if they are **matched** based on the value in their bit-position.

   Detectors are matched only if they have a 1 in their bit position.

2. Detectors notify their immediate superior groups whether they are `matched` or `unmatched`. Each notified group passes a notification to its superiors indicating whether it is `matched` or `unmatched` until all groups have received a notification. A join group is only `matched` if both its subordinate groups are `matched`, otherwise it passes on the notification that it is `unmatched`.

3. Groups are `matched` this cycle and are `unsuspended` send a message to their subordinates which sets them to `support suppressed`. This message is passed down the network

`support suppressed`.

This step ensures only groups at the top of the hierarchy control system behaviour. These groups turn off the lower groups by sending suppression messages down the hierarchy, but only if the higher groups have had sufficient trials to determine if they are useful (i.e if they are unsuspended).

4. Groups which are `matched` and are not `support suppressed` this cycle **create nodes** to predict them for any groups which were `matched` and not `support suppressed` in the previous cycle.

   Create nodes in the groups which controlled behaviour in the previous cycle to predict groups controlling behaviour this cycle. This connects hierarchies together temporally.

5. Groups which are `matched` in this cycle retrieve the system `reward` from the environment and **pay** their predicting nodes a `return` based on the `reward` and the maximum `value estimates` of nodes in the group.

   Groups which controlled behaviour in the previous cycle update their values based on the values of groups which will control behaviour this cycle.

6. Nodes which executed in the previous cycle **update** reinforcement `value estimates` and `transition estimates` (ETPs) based on the immediate reward and `returns` they received from their prediction. If the group they predicted was not `matched` this cycle, the return is set to zero.

7. Groups which are `matched` this cycle, are not `support suppressed` and have more than one predicting group which was not `create suppressed` in the previous cycle, is `unsuspended` and contains nodes which executed in the previous cycle, select two of these predicting groups. They **create a new group** for the selected two groups which is a composite (join) of the two and which contains a node predicting the `matched` group. The new node supports the effector selected in the previous cycle. (*Groups which have one or more predicting nodes which executed in the previous cycle whose ETP exceeds a minimum threshold skip this step.*)

   Create new join groups by selecting two groups that controlled system behaviour in the previous cycle and creating a new join group with these as subordinates. Only groups which are controlling behaviour this cycle create new groups, so we are creating and connecting groups at the top of the respective hierarchies.

8. Groups which contain a node which has reached the minimum `activation criteria` set themselves to `unsuspended`.

   Groups at the top of hierarchies which were undergoing initial evaluation trials to determine their value set a flag to indicate they have completed those trials.

9. Groups which are `unsuspended` and whose nodes all have a lower `ETP` than the equivalent nodes (nodes with the same prediction and effector) in their immediate subordinates **remove** themselves along with all their superiors.

   compare their value with their subordinates. If the values of all the nodes in the superior group are lower than that of the immediate subordinate groups (determined using the Cox-Stuart tests), the superior group is removed.

10. Groups which are `matched` this cycle send notification to their subordinate groups setting them and all their subordinates to `create suppressed`. Groups which were `create suppressed` in the previous cycle set themselves to not `create suppressed` prior to this notification..

    Ensure groups which are **not** *at the top of controlling hierarchies now are not used as subordinates when creating new join groups in the next cycle.*

11. Nodes in `unsuspended` groups which were not `support suppressed` and were `matched` this cycle **fire** and send `support` for their associated effectors.

    Nodes in groups at the top of matched hierarchies control system behaviour

12. The system selects an effector based on the `support` sent by nodes this cycle.

13. Nodes which supported the selected effector **execute**. Groups which were `support suppressed` this cycle record this in another variable in preparation for possibly being `support suppressed` next cycle.

    Allow subordinate groups to be eligible to control behaviour in the next cycle if no superiors are matched in that cycle. Remember that these groups did not control behaviour so that they will not be used in connections next cycle

14. **Repeat**

Acknowledgements

The breast cancer database was obtained from the University of Wisconsin Hospitals, Madison by Dr. William H. Wolberg. Many thanks to Ann Nicholson and David Albrecht for their constructive comments and reviews of this material and also to Dr Selby Markham.

References

Bennett, K. and O. Mangasarian (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software 1*, 23–34.

Buntine, W. and T. Niblett (1992, January). Further comparison of splitting rules for decision-tree induction. *Machine Learning 8*(1), 75–85.

Chapman, D. and L. Kaelbling (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the 12th International Conference on Artificial Intelligence*, Volume 2, pp. 726–731. Morgan Kaufmann.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI-92*.

Daniel, W. (1990). *Applied Non-parameteric Statistics* (2nd ed.). Boston: PWS-Kent.

candidate principle for ethology. In P. Bateson and R. Hinde (Eds.), *Growing Points In Ethology*, Chapter 1, pp. 7–54. Cambridge University Press.

Drescher, G. (1991). *Made-Up Minds: A constructivist approach to Artificial Intelligence*. MIT Press.

Fahlman, S. (1988). Faster-learning variations on back-propagation: An emperical study. In *Proceedings of the 1988 Connectionist Models Summer School*. Morgan-Kaufmann.

Fahlman, S. and C. Lebiere (1988). The cascade correlation learning architecture. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. In *Annals of Eugenics*, Volume 7, pp. 179–188. Cambridge University Press.

French, R. (1999). Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. *Trends in Cognitive Sciences 3*(4), 128–135.

Henery, R. J. (1994). *Machine Learning, Neural and Statistical Classification*, Chapter 2, pp. 6–16. Ellis-Horwood.

Holland, J. (1975). *Adaption in natural and artificial systems*. University of Michigan Press.

Holland, J., K. Holyoak, R. Nisbett, and P. Thagard (1986). *Induction. Processes of Inference, Learning and Discovery*. The MIT Press.

Holte, R. (1993). Very simple classification rules perform very well on most commonly used datasets. *Machine Learning 11*, 63–91.

Kaelbling, L., L. Littman, and A. Moore (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4*, 237–285.

Lin, L. (1993). *Reinforcement Learning for Robots Using Neural Networks*. Ph. D. thesis, School of Computer Science, Carnegie Mellon University, Pittburgh USA.

Littman, M., A. Cassandra, and L. Kaelbling (1995). Learning policies for partially observable environments: Scaling up. In A. Preiditis and S. Russel (Eds.), *Machine Learning: Proceedings of the 12th International Conference*, pp. 362–370. Morgan Kauffman.

McCallum, A. (1995). *Reinforcement Learning With Selective Perception and Hidden State*. Ph. D. thesis, Department of Computer Science, University of Rochester, NY.

McCallum, A, R. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the tenth international machine learning conference*.

McCloskey, M. and N. Cohen (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation 24*, 109–164.

Michie, D., D. Spiegelhalter, and C. Taylor (Eds.) (1994). *Machine Learning, Neural and Statistical Classification*, Chapter 9. Ellis Horwood.

Mitchell, M. (2000). Combining classification and temporal learning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence*, Austin, Texas. AAAI: AAAI Press/The MIT Press.

Hill.

Peng, J. and R. Williams (1993). Efficient learning and planning within the dyna framework. In J. Meyer, H. Roitblat, and S. Wilson (Eds.), *From Animals to Animats 2*, USA, pp. 281–290. Second International Conference on Simulation of Adaptive Behaviour: MIT Press.

Ring, M. (1994). *Continual Learning in Reinforcement Environments*. Ph. D. thesis, The University of Texas at Austin.

Sutton, R. (1991). Reinforcement learning architectures for animats. In J. Meyer and S. Wilson (Eds.), *From Animals to Animats*, pp. 288–296. First International Conference on Simulation of Adaptive Behaviour: MIT Press.

Tan, M. (1991). Learning a cost sensitive internal representation for reinforcement learning. *Proceedings of the 8th International WorkShop on Machine Learning*, 358–362.

Thrun, S. et al. (1991, December). The MONK's problems. A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

Watkins, C. (1989). *Learning From Delayed Rewards*. Ph. D. thesis, Cambridge University.