

Learning Prediction-Models in Environments with Hidden-state

Matthew Winston Mitchell
Monash University, Australia
matt@csse.monash.edu.au

Abstract

TRACA (Temporal Reinforcement-learning and Classification Architecture) is a new reinforcement learning system which incrementally develops a rule-based predictive model of its environment while learning. In particular, TRACA is designed for sequential decision problems in large state spaces which contain hidden-state and in which both sensor and effector noise are present. It is one of only a few systems (another is U-tree) intended for the types of problems which are encountered by learning robots situated in unstructured real-world environments.

TRACA combines three capabilities: input generalisation, sequential chaining and short-term memory. Each of these have previously been demonstrated on common problems, including one experiment which demonstrated TRACA's potential for on-line planning.

This paper presents further experiments based on a simple, but representative, spatial navigation domain. This domain is used to demonstrate two capabilities of TRACA which are absent from similar systems such as U-tree. Finally, the paper highlights one significant issue which needs to be addressed before further progress can be made using systems which deal with hidden-state while performing input generalisation.

1 Introduction

TRACA is a relatively new system developed as a sub-cognitive architecture for situated learning agents operating in real-world environments [Dorigo and Colombetti, 1994]. One problem faced by learning agents in real-world environments is the size of the state space. Large state spaces can be dealt with using input generalisation techniques which map the large number of

world states to a smaller number of internal states [Chapman and Kaelbling, 1991]. Input generalisation is also commonly used in the fields of classification and concept learning. In previous work TRACA has been demonstrated to compare favourably to a number of common systems on a range of input generalisation tasks in terms of predictive performance and solution size [Mitchell, 2003a]. Sequential decision problems, such as robot navigation and animat problems, require also the ability to chain together generalised rules across time [Wilson, 1987]. This can lead to a combinatoric explosion as the number of rules in the system grows [Chapman and Kaelbling, 1991]. However, if tractable chaining of rules can be achieved, a number of benefits are promised including latent-learning, on-line planning and accelerated reinforcement learning [Riolo, 1991; Whitehead, 1989].

One common approach to sequential decision problems in large state spaces is to use Holland Style Learning Classifier Systems (LCSs) [Holland, 1975]. These systems often rely upon a genetic algorithm (GA) to learn a set of rules and GA driven classifier systems have proven successful at performing input generalisation [Saxon and Barry, 1999]. However, the GA has been less successful in dealing with sequential decision problems and in practice it has proven necessary to supplement the GA with a number of additional rule creation operators [Riolo, 1989; 1991; Stolzmann, 2000].

Motivated by the LCS approach, TRACA has been designed to address the issues of performing chaining and input generalisation for sequential decision problems. In place of a GA, TRACA uses a variety of heuristic search techniques which incorporate selective mechanisms [Foner and Maes, 1994]. TRACA's rule structures are similar to the schemata created by the Schema mechanism of Drescher [1991], however, TRACA uses more efficient default hierarchies [Holland *et al.*, 1986] along with a suppression mechanism to remove subordinate and equivalent structures from the combinatorics of search and chaining. A further difference is that TRACA

creates new components by combining existing components.

Unlike standard Holland Style classifier systems, TRACA has an inbuilt mechanism to deal specifically with *hidden-state* (also known as perceptual aliasing or Partially Observable Markov Decision Problems) [Colombetti and Dorigo, 1994; Whitehead and Lin, 1995]. Cliff and Ross [1995] added temporary memory capabilities to a Holland Style classifier system and demonstrated that this memory could be used successfully if the number of bits required is small, but their approach does not scale well. Robertson [1988] used triggered chaining operators in Learning Classifier Systems to create chains to predict letters in a sequence that required remembering earlier characters in the sequence. While it could do this successfully for some simple sequences, it failed for more difficult sequences. Many other non-classifier based approaches have been used for hidden-state. Often these either side-step the input-generalisation problem (for example, Schmidhuber [1997] who uses Levin search and Wiering [1997] who uses HQ learning) or suffer from scalability problems (for example, recurrent neural networks [Lin and Mitchell, 1992; 1993]).

One approach which does deal with both input-generalisation and hidden-state is McCallum’s [1995] U-tree. TRACA and U-tree are similar in their approach to hidden-state. However, TRACA deals with two important issues which U-tree does not. The first of these issues is the ability to deal with homogeneous regions of hidden-state. The second, is the ability to perform chaining efficiently given the presence of irrelevant features. Both of these issues are described in detail in the following sections.

The next section describes TRACA and its approach for representing hidden-state. Following this, homogeneous hidden-state regions are discussed and some experiments are presented which demonstrate the effects homogeneous regions have on learning times. Finally, an experiment is presented in which irrelevant attributes are introduced along with an associated technique to direct learning appropriately.

2 System Description

TRACA is an incremental reinforcement learning system which receives environmental inputs using a fixed length bit string. Feedback on its performance is provided as a scalar reinforcement signal [Sutton and Barto, 1998]. TRACA operates in discrete time, in which an observation is received and an action is taken, this is followed in the next timestep by the observation of the resulting state and so on. In the following discussion it is assumed that a localised representation scheme is used where each possible observation in the environment is mapped to a

unique bit in the input string. To represent actions the system has an array of effectors, one for each possible action.

2.1 Low-level Structures

When inputs are received from the environment, each bit in the input string is mapped to a low-level internal state representation called a *unary group*. When a localised representation scheme is being used, unary groups represent observations. If the unary group’s bit position in the current string contains a 1 the group is *matched*, if it contains a 0 it is not matched. Groups may also contain a number of nodes, each of which represents a situation-response-situation (SRS) rule. Each node has an associated action and a prediction. Nodes store estimated transition probabilities (ETPs) to other groups (their *predictions*) given that their group is matched and their action is selected. Nodes also maintain a utility estimate of the value of their action given that their group is matched. This estimate is calculated based on Q-learning [Watkins and Dayan, 1992]. Both estimates are recency-weighted. Nodes in matched groups can influence which action the system selects by sending their utility value estimate to the effector array in support of their action. The system can then select an action based on the effector which received the highest support or, with a small probability, the system may take an exploratory action in which case the action is selected randomly.

The group and node structures just described are not sufficient to represent problems which contain hidden-state. For hidden-state problems additional memory structures are required. TRACA implements the required memory using *temporal chains*. Each link in a temporal chain is a *temporal group* (a grouping of temporal rules). Temporal chains are constructed incrementally over a number of trials in the environment. When using a localised representation scheme, the initial temporal group in a chain represents the matching of two unary groups in immediate succession. But each subsequently added temporal group will extend the chain by one additional unary group (i.e one discrete timestep). Hierarchically, each temporal group is *superior* to the two *subordinate* groups used in its representation. A temporal group is matched once both its subordinates have been matched in succession. The entire chain is matched once the path covered by the chain has been followed, at which point the *terminal node* will update its ETP based on the presence of its predicted observation (terminal nodes are described in Section 2.3). Temporal chains are constructed incrementally during learning to include the history necessary to uncover one hidden-state (perceptually aliased) region.

2.2 Temporal Chain Construction

The process of constructing temporal chains consists of three tasks: chain creation, chain extension and search restriction.

Temporal chain creation occurs during learning trials in the environment when a condition is identified which may indicate the environment contains hidden-state. Each learning trial consists of a number of timesteps. In each timestep an observation is presented to the agent and the agent selects an action which may affect the observation received in the following timestep. Trials commence with the agent being placed in one of the possible initial states and conclude once it reaches a terminal state, at which point the agent is removed from the environment.

In deterministic environments with a localised representation scheme, hidden-state can be identified when a rule (node) incorrectly predicts a unary group (i.e the rule incorrectly predicts an observation). Table 1 illustrates a deterministic environment which contains hidden-state. The table shows possible sequences of observations received by an agent during learning trials in this environment. In this domain the agent has only one possible action which always leads it to the next state in the sequence. At the start of each trial the learning agent is placed in one of the equally probable initial states and receives an observation which is one of D, E or F. The system then executes the action which places the agent in the next state which always results in observation A. Executing the action when the observation A is experienced leads to one of the three terminal states and either observation X or observation Y depending on the initial observation. To accurately predict the third observation the agent must remember whether the first observation was D, E or F.

In the environment in Table 1, memory of just one previous state and action is sufficient to make accurate predictions, however, in other problems memory may need to extend further back in time. The process of adding additional history, or memory, to a created chain in an attempt to predict a state reliably is called *chain extension*. Additional history is added incrementally, with each new addition being created for a particular group and action. Ideally, extension continues until the temporal chain includes enough history to reliably predict the state it was created to predict. Since k previous observations are used to make reliable predictions, TRACA uses a Markov- k approach to handling hidden-state, U-tree is another Markov- k algorithm. Both U-tree and TRACA allow k to vary in different areas of the search space.

In reality it is possible that no amount of history will improve the predictions of the chain. This problem may arise if there are stochastic processes in the environment. For example, if B represents standing on the sidewalk

and A the passing of a car, no amount of history (within the percepts of the agent) will help predict A. It is in situations like this that search restriction is necessary. One method of restricting search is to maintain a fixed sized history window with a predetermined size to prevent endless searches. However, if the window size is too small, necessary hidden-state will not be uncovered, similarly, if it is too large, resources may be wasted. TRACA uses an approach which allows the maximum length of chains (i.e the window size) to vary in different areas of the search space. This sizing of the window is based upon estimates of the benefits of constructing a chain to represent a particular aliased region in the environment. The benefits are calculated using the estimated returns associated with the observation predicted by the chain under construction.

t_1	t_2	t_3
D	A	X
E	A	Y
F	A	Y

Table 1: A sample deterministic environment with hidden-state.

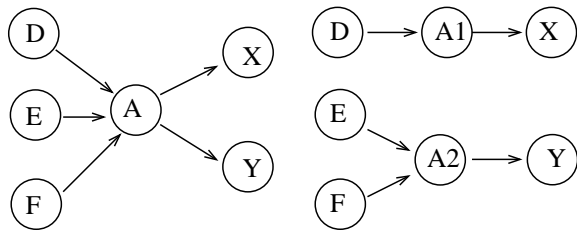
2.3 Problem Search Space

The search to uncover hidden-state may be represented using a directed graph of observations and action transitions. Given a particular observation, the graph forms a tree, the root is the observation to be predicted, the vertices correspond to possible prior observations and the edges correspond to actions leading from one observation to another. Each branch in the tree represents sequences of states and actions which form paths from the states represented by the leaves to the state represented by the root. Since we are interested in searching this tree for appropriate paths from branch states to the tree root, the tree is referred to as the *search-tree*. The size of the search tree may increase exponentially as it is expanded down all possible branches.

When dealing with hidden-state, the observation we are constructing a chain to predict is the tree root. During chain extension, the chain created so far is the *search-path*, which continues to be extended in depth down the tree until either the hidden-state is uncovered, in which case the path is *complete*, or the search is terminated by restrictions on search size. The algorithm for creating chains involves a search through this space of observations and action sequences. For each search tree the group representing the tree root is the *chain prediction*. The temporal group reached first when following the chain path from its leaf to the root is the *first group* and the group reached last is the *terminal group*. Within

the terminal group the node which predicts the chain prediction is the *terminal node*.

When a domain contains hidden-state, the search-tree includes paths which do not form valid walks in the graph to our root. This is because the graphs (or sub-graphs) appear connected until shared vertices are distinguished as separate environment states. Our example presented in Table 1 appears to be a connected graph (Figure 1(a)), whereas it is in fact a disconnected graph (Figure 1(b)).



(a) A connected graph representation of the domain presented in Table 1.

(b) Once hidden state is distinguished the connected graph in (a) appears as a disconnected graph. In this case, the vertex A has been split into two vertices A1 and A2 in two separate trees.

Figure 1: Two graphs before and after hidden-state is uncovered.

Because the shared vertex is not distinguished as we are searching our tree, search-paths can be formed representing walks which can never actually occur in the problem domain. Figure 2 shows the valid and invalid paths that can be formed in a tree structure which has been expanded from the root down.

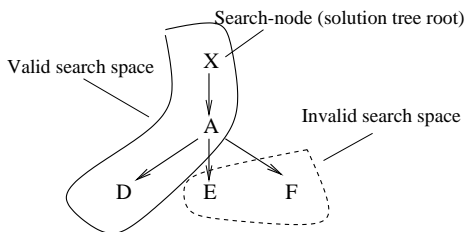


Figure 2: Valid and invalid search spaces in the solution tree for X in the problem from Table 1.

2.4 The Search Strategy

The search strategy aims to discover a valid search-path which leads to the root. It is perhaps best described as a depth-limited iterative depth-first search. That is to say, that starting from the root, the search will extend down an arbitrary branch to the leaf. If an invalid path is followed, the most recent extension is removed and another created down a different branch. The depth is limited by the utility value of the chain-prediction representing the root and if the search extends too far, the entire path created so far is removed and another search commenced.

These searches are conducted during actual trials, where each trial provides an opportunity for at most one extension for each chain (i.e searches may execute concurrently for different chains). As such, chain creation requires a number of walks (and possibly a variety of walks) being taken through our graph. In each walk once the observation corresponding to the chain's current first group is met, the search-path (and the chain representing it) can be extended to the most recently visited vertex which is equivalent to extending it down one branch of the search tree. If we continue up the new search-path and find that we do not follow a valid path to the root the search-path has been extended to form an invalid path and needs to be back-tracked.¹ Using temporal groups to store information on the observations and actions experienced during a walk, chains can keep track of whether their path is being followed and only evaluate whether the new extension node is valid once their path has been completed. In the case of an invalid path, the most recent extension is removed and we wait for another trial in which the solution path is extended again, hopefully (due to exploratory actions) to a sibling node of the removed node which forms a valid path. This is the *iterative* component and in its simplest form is not systematic, no record is kept of which paths and siblings have been tried (Section 7 describes a method which improves this situation). Given this, if a valid extension is found we proceed in a depth-first manner until the chain is complete. Note also, that due to randomness there is no guarantee that any particular branch will be followed first.² The maximum depth (solution path length) it will extend to is restricted based on the reinforcement associated with the state corresponding to the root. The value of the prediction is passed back along the chain and discounted at each link. If by the time it reaches the first group it falls outside a threshold value, the entire chain is removed.³

¹This requires making sure you follow the search-path as if you deviate you cannot determine if you reach the root.

²This prevents the use of a more efficient systematic search method such as depth-first iterative deepening [Korf, 1987].

³Different thresholds are applied for positive and negative

3 Types of Hidden-state

This section presents two example environments which exemplify two different types of hidden-state that may occur. The example environments discussed are both simple mazes with an East and a West branch. Each maze has boundaries which are impassable walls and at the end of each East branch is a door. The maze is divided into discrete states with an input string representing each state’s features. Possible features are the presence of windows, doors and any immediate surrounding walls. An agent navigating in this simplistic environment may select an action which moves it from it’s current state to one of its neighbour states, unless the action leads the agent into a wall, in which case the agent’s state will remain unchanged. Possible actions are Move-south, Move-north, Move-east and Move-west. Before selecting each action, the agent receives an input string indicating the features of its current state.

The first maze is depicted in Figure 3. The East branch of this maze provides an example of the first type of hidden-state called a *heterogeneous* region. The region begins in the north-east corner and an agent moving south from that corner will experience a sequence of different observations up until the south-east corner. It contains hidden-state because the sequence of observations received while traversing the region is shared by some other region in the state space, in this case, by the West branch. It is called heterogeneous because each state within the shared region of observations has a different input string. As each feature is represented separately in the input string, this example is using a *distributed* rather than a *localised* representation scheme.

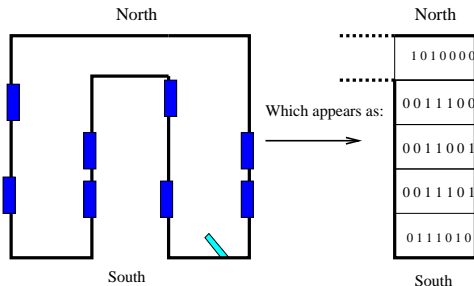


Figure 3: A region (corridor) with a heterogeneous input pattern.

The maze in Figure 4 illustrates the second type of hidden-state. This maze contains a homogeneous region of hidden-state. Once again, the region is the set of states traversed by an agent commencing at the north-east corner and continuing south until it reaches the

south-east corner. Again, the region contains hidden-state because a sequence of observations received while traversing the region is shared by some other region in the state space (again by the West branch). However, in a homogeneous region such as this, there is an additional dimension to the hidden-state. The same observation appears for multiple states within the hidden-state region.⁴

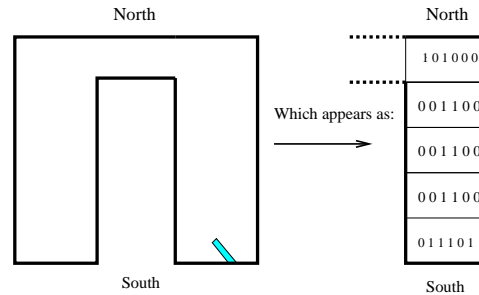


Figure 4: A region (corridor) with a homogeneous input pattern.

This distinction between homogeneous regions and heterogeneous regions is critical, as homogeneous regions need to be treated differently during learning.

4 Representing Homogeneous Regions

There are two major problems associated with constructing Markov-*k* chains to represent homogeneous regions of hidden-state.

The first problem is detecting when the chain provides an improvement over existing structures. In the case of chains representing heterogeneous regions this can be done by comparing the terminal node’s ETP to the ETP of the node in its subordinate group which has the same action and predicts the chain prediction. If, after a suitable number of trials, the terminal node provides an improvement over the subordinate the chain can be retained, otherwise it should be removed (in fact, in TRACA the comparison is based on a non-parametric test for significance which allows for non-deterministic environments. See [Mitchell, 2002] for details).

However, for a chain representing a homogeneous region real improvements cannot be detected so easily. This is because the greater specificity of a chain extending across a homogeneous region can make it appear more useful than it really is. The terminal node in any chain which extends over two or more positions will

⁴When a distributed representation scheme is being used, then depending on the path taken by the agent down these corridors, and the information used by the agent to discriminate positions, each of these corridor types may appear to be a case of the other type.

have a higher ETP than a subordinate structure which includes only one. A simple solution in the case of homogeneous chains is to require that the ETP of the terminal node exceed a threshold value (typically close to 1).

The second problem is due to the process of chain extension. Chains which are due to be extended add a previous observation as additional history when the search path completed so far (the first group) is reached. The extension is then retained only if the agent continues to traverse the chain's path and that traversal leads to the chain prediction.

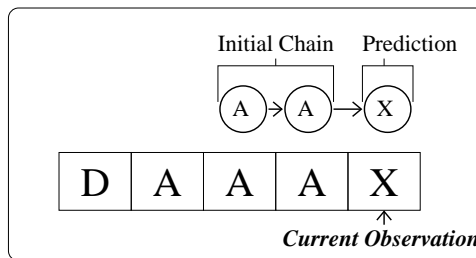
For heterogeneous regions and for homogeneous regions extending over only two aliased states this presents few problems. However, for homogeneous regions extending over more than two aliased states there are additional complexities which only arise during the extension of chains (and not once a chain is correctly completed). The problem is that the search-path of partially constructed chains may match multiple different sub-regions of the aliased region it is being build to represent. In this scenario it is possible that a partially constructed chain never leads to the chain prediction when its path is followed.

To demonstrate this point consider the prediction problem presented in Table 2. This task is similar to the one discussed in Section 2.2. There are now two possible sequences of observations, one beginning with D and terminating with X the other beginning with E and terminating with Y. The initial states D and E are equiprobable and in each state the agent has a single possible action. In both sequences, the region between the initial letter and the terminating letter is a perceptually-aliased homogeneous region of three consecutive A's.

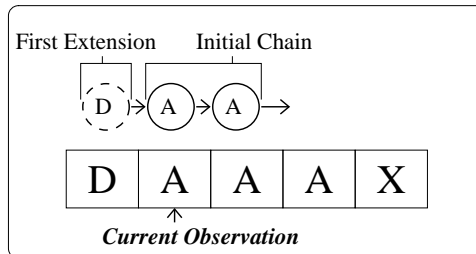
t_1	t_2	t_3	t_4	t_5
D	A	A	A	X
E	A	A	A	Y

Table 2: A problem with two possible sequences of observations.

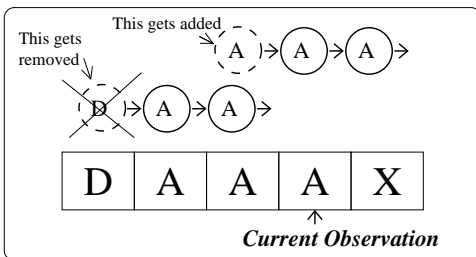
Figure 5 demonstrates various stages in the construction of a chain to correctly predict X. The first stage is the initial construction of the chain. In this stage, the chain covers the search path containing the state prior to X and its immediate predecessor state (see Figure 5(a)). In the second stage, the agent experiences another presentation of the sequence beginning with D. The first A in this sequence matches the first-group, so the chain is extended to include D (Figure 5(b)). However, continuing the sequence with the subsequent two presentations of A leads to the removal of the extension to D. This removal is due to the discovery that following the newly extended chain path does not lead to the chain prediction



(a) Initial creation of the chain for the D sequence.



(b) Extension of the chain for the D sequence.



(c) Removal of the first extension and an attempt to create another.

Figure 5: Three different stages of chain construction and extension for a sequence to represent the path from D to X.

(observation X) but rather the observation A. However, the occurrence of this third A again matches the first group of the chain (since the chain's path is no longer being followed) and a new extension is created to include the prior observation, which in this case was the second A in the sequence (see Figure 5(c)). However, the final observation experienced is X, not A, so the chain's path has not been followed and this most recent extension is also removed.

Without a modification to the rules for extending chains, the current chain will never be extended correctly

and the aliased regions never distinguished. The necessary modification is to allow a chain extension to be retained not only if the chain prediction occurs subsequent to following the chain path, but also if the observation associated with the prediction is experienced in place of an observation expected if the currently represented chain path is being followed. This allows the extension made in Figure 5(c) to be retained and the final extension to D to be made and retained in a subsequent trial.

5 Experimental Methodology

TRACA has already been evaluated on a range of hidden-state problems from the literature (see [Mitchell, 2003b]). Therefore, the first set of experiments presented here focus specifically on the differences between representing homogeneous and heterogeneous spatial regions. The second experiment moves away from our localised sensor scheme, to a distributed sensor scheme. Under a distributed sensor scheme different features in the environment are represented by different bit positions in the input string. Furthermore, many different features may be present at the same time. This second experiment demonstrates an approach to guiding search when one or more of these features is less useful (eg. completely irrelevant) to the current task the agent is performing and therefore should be ignored when searching for appropriate history features on which to construct memory.

6 Gap Tasks

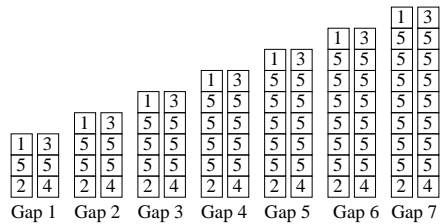
Mozier [1992] introduced gap tasks to test his *Reduced Description Network*. Success on these tasks required the system to remember an initial observation across a fixed sequence of observations in order to correctly predict the final observation in the sequence. Ring [1994] also tested his *Temporal Transition Hierarchies* on these tasks, and more recently Bakker [2002] has tested LSTM networks on T-maze problems similar to Mozier’s gap tasks.

The gap tasks here serve two purposes. The first is to demonstrate that TRACA can remember observations for varying lengths of time. Gaps of up to 7 observations have been included in the experiments. These tests are sufficient to demonstrate that TRACA can represent gaps of arbitrary length, as long as a single path to the goal is followed with sufficient frequency.

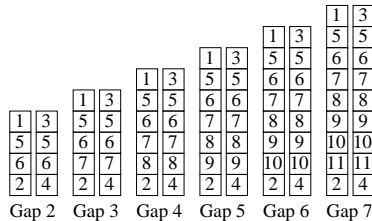
The second purpose of the gap tasks, is to compare the learning times for sequences of observations (corridors) that are homogeneous and heterogeneous.⁵ In the following experiments with TRACA the gap problems are represented as simple maze navigation tasks. The homogeneous tasks are depicted in Figure 6(a) and the

⁵Mozier [1992]’s original experiments did not include homogeneous gaps, therefore the difficulties they present did not arise.

heterogeneous tasks in Figure 6(b). Each gap experiment involves one maze with two possible corridors with a fixed length gap. In Figures 6(a) and 6(b) the numbers at the top of each task are the observations seen at the start of each sequence and the numbers at the bottom are the observations seen at the end of each sequence. The system must remember the observation at the start of a sequence in order to reliably predict the observation at the end.



(a) The six homogeneous tasks and the Gap 1 task.



(b) The six heterogeneous tasks.

Figure 6: The mazes used in the gap experiments. In the homogeneous tasks the same observation is repeated across the shared region. In the heterogeneous tasks each observation is different within the shared region.

6.1 Experimental Design

Each gap experiment consists of a single training episode with a number of trials. At the start of each trial the agent is placed at the top of one of the corridors (selected with equal probability). The agent can select from four possible actions, Move-Up, Move-Down, Move-Left, Move-Right. In positions other than those at the top and bottom of corridors, selecting Move-Up will place the agent in the position above its current position, selecting Move-Down will place it in the position below. In all positions selecting Move-Left and Move-Right will leave the agent’s position unchanged, as will selecting Move-Up in a top position. The position at the bottom of each corridor is the goal. On reaching the goal the agent receives a positive reward of 100 and the trial terminates. No other rewards or penalties are provided. The experiment is completed once the agent has developed all the structures necessary to disambiguate each position in each corridor. For each task, two chains are required, one to represent each corridor. For example,

the Gap 1 task in Figure 6(a), requires a chain to remember the initial observation 1 across the hidden-state region when observation 5 is presented in order to reliably predict observation 2. A similar chain is needed to predict observation 4 when the sequence started with observation 3. Construction of the required chains is confirmed using a manual inspection of the structures developed and retained in the system, guided by a visualisation tool which indicates when temporal chains have been completed.

Node values are updated with only immediate rewards (penalties) if they predict their containing group (since such situations should eventually be avoided by the creation of temporal chains). Therefore, the probability of selecting actions to move into walls in the gap tasks depends on the exploration support given to actions with zero value. In these experiments this value is 0.5. With probability 0.1 an action is selected using Golberg’s [1988] roulette wheel approach, otherwise if one effector receives higher support than another it is selected. A learning rate of 0.2 is used.

6.2 Results

Table 3 shows the number of timesteps that elapsed during training trials on each of the gap tasks. For the task with a gap of 1 it took the agent 309 steps in the environment to learn the two temporal chains required. These steps will have included a number of exploratory moves into walls and along a variety of paths to the goal. Before chains can be created and support sent to effectors to drive the agent down the shortest path to the goal, a number of trials were needed for each of the nodes in the unary groups to develop ETPs and utility estimates. Learning times for homogeneous gaps increase linearly from 309 for the Gap 1 task up to 1339 timesteps for a gap of 7.

The heterogeneous gap tasks take up to 3 times longer to learn than the homogeneous tasks with the same gap size. The heterogeneous learning times range from 535 timesteps for a gap of 2 up to 3902 for a gap of 7. These differences between learning times for homogeneous and heterogeneous mazes increase in proportion to gap lengths. The reason for this is that in homogeneous corridors, unary nodes in the hidden-state regions are executed more frequently as the observation matching their group is presented more often. This leads to the agent sending support and returns sooner which leads it along paths to goals sooner and more frequently.⁶

⁶In deterministic environments such as our gap tasks, the number of learning trials could be improved by reducing the number of trials required by unary nodes before relying on their ETPs and sending their utility estimates as support. However, this will not generalise to non-deterministic environments.

Gap	Steps to Learn	
	Homogeneous	Heterogeneous
1	309	n/a
2	330	535
3	469	1046
4	636	1756
5	861	2166
6	1015	3116
7	1339	3902

Table 3: Comparative performance on the homogeneous and heterogeneous gap tasks.

Of interest in the gap tasks is the scenario of changing the reward function to make it equally desirable to go both ways in the hidden-state regions (this is related to the problems described by Whitehead and Ballard [1991] where learning may diverge from the optimal policy). Consider a variation on the gap tasks for the homogeneous gap 3 task in Figure 6(a) where the only reinforcements provided are equal positive rewards for reaching positions 1 and 4. At the start of each trial the agent is placed in one of the positions 2 or 3 with uniform random probability. Once initial trials for nodes are complete and rewards have been received for each corridor, the agent oscillates between one direction and the next. Chains could overcome this problem, but the creation of chains is prevented by the agent’s oscillating behaviour which rarely traverses paths to either goal. One generic solution suitable for spatial navigation tasks depends on choosing an appropriate representation of the state space [Mitchell, 2003b].

7 Irrelevant Noisy Features

In the real world, people move in and out of corridors. The presence of people in the corridor is both unpredictable and irrelevant to navigating the corridor (assuming they keep out of the way). Ideally we would like our chains to be extended back using features other than the presence of a person. Unfortunately, the use of values which measure the predictive accuracy of a node, such as our ETPs, are useless for this selection as they are calculated in isolation of other features (i.e nodes in other groups). Instead, we must introduce a new measure which compares different predictors and selects the one which most frequently precedes our prediction. This new measure is referred to as the *precedence rate* and it is calculated not by the predicting node, but the predicted group. The next experiment demonstrates the use of the precedence rate to direct the extension of chains towards the most reliable features.

7.1 Experimental Design

This experiment is based on the homogeneous gap 5 task from Section 6. However, now a distributed sen-

sors scheme is used so that the presence of a person in the same location as the agent (in any aliased corridor state) can be indicated by a value of 1 (and 0 otherwise) in a unique bit position in the agent’s input string. The input string also has a unique bit for each of the corridor location observations from the original Gap 5 task. The problem still contains hidden-state, except now for each of the corridor locations labelled 5 in the gap 5 task of Figure 6(a) the probability of a person being present is 0.25. All other parameters and experimental conditions are unchanged from Section 6. Learning is stopped once chains have been constructed which remember the initial observation and accurately predict the two goal states when the shortest path is followed from the initial state. Therefore, one chain must extend from the position labelled 1 to the position labelled 2, the other from the position labelled 3 to the one labelled 4. The presence of the required chains is determined using a visual tool to guide manual inspection of TRACA’s structures as they are completed. Five agents were trialled on this task, each with a different random seed.

7.2 Results

TRACA developed the required chains with an average of 871 (standard deviation 97.6) timesteps in the maze. This learning time is not substantially different from the same task without the person of 861 (see Section 6). For every chain created, no links were constructed on the group (bit position) indicating the presence of a person. A similar trial which selected groups for chain extension based on their dependent ETP failed to construct chains which excluded extensions built upon the feature indicating the presence of a person.

The creation of only two chains for each run on this task provides a large efficiency in learning and representation. An approach using localised sensors would require chains for every possible combination of features that could be traversed in the hidden-state region (assuming the best case scenario where the shortest path to each goal is always followed).

When using distributed sensors, the use of a precedence rate provides a powerful bias to extend chains down paths which most regularly lead to predictions. In this case, it allows TRACA to avoid irrelevant noisy features in such searches.

8 Conclusion and Future Work

This paper has identified two important difficulties which arise when constructing predictive Markov- k models in environments which contain hidden-state. The first difficulty is dealing with homogeneous regions, the second with irrelevant noisy features. The paper then described approaches to deal with each of these difficulties and

demonstrated these approaches experimentally on simple, but representative, domains. A project is currently being undertaken to demonstrate TRACA’s approach using a real Nomad Scout mobile robot. However, one important open problem for Markov- k approaches is obtaining generality when performing input generalisation in the presence of multiple features. The problem is that temporal chains do not generalise when there are multiple structures representing the same concept. For example, a decision tree representation (such as used by U-tree) may have many leaves representing the concept *chair*. However, current Markov- k approaches will construct a temporal chain that extends across only one of these leaves, even though for generality all leaves representing the concept should be considered equivalent. It is possible that TRACA may be able to achieve this generality by considering the predictions of the groups subordinate to chain links and treating those with common predictions as equivalent.

References

- [Bakker, 2002] B Bakker. Reinforcement learning with long short-term memory. In T.G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1475–1482, Cambridge, MA, 2002. MIT Press.
- [Chapman and Kaelbling, 1991] D Chapman and L.P Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, volume 2, pages 726–731. Morgan Kaufmann, 1991.
- [Cliff and Ross, 1995] D. Cliff and S Ross. Adding temporary memory to ZCS. *Adaptive Behaviour*, 3(2):101–150, 1995.
- [Colombetti and Dorigo, 1994] M Colombetti and M Dorigo. Training agents to perform sequential behaviour. *Adaptive Behaviour*, 2(3):247–275, 1994.
- [Dorigo and Colombetti, 1994] M Dorigo and M Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71:321–370, 1994.
- [Drescher, 1991] G.L Drescher. *Made-Up Minds: A constructivist approach to Artificial Intelligence*. MIT Press, 1991.
- [Foner and Maes, 1994] L.N Foner and P Maes. Paying attention to what’s important: Using focus of attention to improve unsupervised learning. In D. Cliff, P Husbands, J Meyer, and S.W Wilson, editors, *From Animals to Animats 3*, pages 256–265. Third International Conference on Simulation of Adaptive Behaviour, MIT Press, 1994.

- [Goldberg and Holland, 1988] D.E Goldberg and J.H Holland. Genetic algorithms and machine learning. *Machine Learning*, 3:95–100, 1988.
- [Holland *et al.*, 1986] J.H Holland, K.J Holyoak, R.E Nisbett, and P.A Thagard. *Induction. Processes of inference, learning and discovery*. The MIT Press, 1986.
- [Holland, 1975] J.J Holland. *Adaption in natural and artificial systems*. University of Michigan Press, 1975.
- [Korf, 1987] R.E Korf. Planning as search: a quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.
- [Lin and Mitchell, 1992] L Lin and T Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, USA, 1992.
- [Lin and Mitchell, 1993] L Lin and T.M Mitchell. Reinforcement learning with hidden states. In J Meyer, H.L Roitblat, and S.W Wilson, editors, *From Animals to Animats 2*, pages 271–280, USA, 1993. Second International Conference on Simulation of Adaptive Behaviour, MIT Press.
- [McCallum, 1995] A.K McCallum. *Reinforcement Learning With Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, NY, 1995.
- [Mitchell, 2002] M.W. Mitchell. An evaluation of TRACA’s generalisation performance. Technical Report 2002/127, School of Computer Science and Software Engineering, Monash University, 2002. Available at <http://www.csse.monash.edu.au/publications>.
- [Mitchell, 2003a] M.W Mitchell. A performance comparison of TRACA - an incremental on-line learning algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1897–1902, Portland, Oregon, USA, 2003. IEEE.
- [Mitchell, 2003b] M.W Mitchell. Using Markov- k memory to represent hidden-state. In *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications*, pages 242–248, Las Vegas, USA, 2003. CSREA Press. Available at <http://www.csse.monash.edu.au/~matt>.
- [Mozer, 1992] M.C Mozer. Induction of multiscale structure. In J.E Moody, S.J Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282, San Mateo, California, 1992. Morgan Kaufmann Publishers.
- [Ring, 1994] M Ring. *Continual learning in reinforcement environments*. PhD thesis, The University of Texas at Austin, 1994.
- [Riolo, 1989] R.L Riolo. The emergence of coupled sequences of classifiers. In J.J Grefenstette, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 256–264, California, 1989. Morgan Kaufmann.
- [Riolo, 1991] R.L Riolo. Lookahead planning and latent learning in a classifier system. In J Meyer and S.W Wilson, editors, *From Animals to Animats*, pages 316–326. First International Conference on Simulation of Adaptive Behaviour, MIT Press, 1991.
- [Robertson and Riolo, 1988] G.G Robertson and R.L Riolo. A tale of two classifier systems. *Machine Learning*, 3:139–160, 1988.
- [Saxon and Barry, 1999] S Saxon and A Barry. XCS and the monk’s problems. In W Banzhaf, J Daida, A.E Eiben, M.H Garzon, V Honavar, M Jakiela, and R.E Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1. Morgan Kaufmann, 13-17 1999.
- [Schmidhuber *et al.*, 1997] J Schmidhuber, J Zhao, and M Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.
- [Stolzmann, 2000] W Stolzmann. Learning classifier systems: From foundations to applications. In P.L Lanzi, W Stolzmann, and S.W Wilson, editors, *Lecture Notes in Computer Science*, volume 1813. Springer-Verlag, 2000.
- [Sutton and Barto, 1998] R.S Sutton and A.G Barto. *Reinforcement Learning: An introduction*. MIT Press, 1998.
- [Watkins and Dayan, 1992] C.J.C.H Watkins and P Dayan. Technical note, Q-learning. *Machine Learning*, 8:279–292, 1992.
- [Whitehead and Ballard, 1991] S.D Whitehead and D.H Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.
- [Whitehead and Lin, 1995] S.D Whitehead and L Lin. Reinforcement learning in non-Markov environments. *Artificial Intelligence*, 73(1-2):271–306, 1995.
- [Whitehead, 1989] S.D Whitehead. Thesis proposal: Scaling reinforcement learning systems. Technical Report 304, Computer Science, University of Rochester, 1989.
- [Wiering and Schmidhuber, 1997] M Wiering and J Schmidhuber. HQ-learning. *Adaptive Behaviour*, 6(2):219–246, 1997.
- [Wilson, 1987] S.W Wilson. Classifier systems and the animat problem. *Machine Learning*, 2:199–228, 1987.