

Hierarchical Traces for Reduced NSM Memory Requirements

Torbjørn S. Dahl

Abstract This paper presents work on using hierarchical long term memory to reduce the memory requirements of nearest sequence memory (NSM) learning, a previously published, instance-based reinforcement learning algorithm. A hierarchical memory representation reduces the memory requirements by allowing traces to share common sub-sequences. We present moderated mechanisms for estimating discounted future rewards and for dealing with hidden state using hierarchical memory. We also present an experimental analysis of how the sub-sequence length affects the memory compression achieved and show that the reduced memory requirements do not effect the speed of learning. Finally, we analyse and discuss the persistence of the sub-sequences independent of specific trace instances.

1 Introduction and Motivation

This paper presents a novel long term memory (LTM) structure that reduces the memory requirements of nearest sequence memory (NSM) learning. NSM is a simple instance-based reinforcement learning (RL) algorithm originally presented by McCallum [14]. While instance-based algorithms learn fast, one of their disadvantages is their relatively large memory requirements. The memory structure we present reduces the NSM memory requirements by allowing the *traces* [20] stored in LTM to share common sub-sequences.

Our aim with this work is to develop efficient RL algorithms that can scale up to complex learning problems such as robot control. We believe this can only happen when a number of properties have been brought together in a single algorithm. In terms of capability such an algorithm must be able to estimate future rewards accurately in the presence of hidden states. In terms of capacity it must, like the

Torbjørn S. Dahl

Cognitive Robotics Research Centre, University of Wales, Newport, Allt-yr-yn Avenue, Newport, NP20 5DA, United Kingdom, e-mail: torbjorn.dahl@newport.ac.uk

human cortex, use both *hierarchy* and *auto-associativity* for efficient encoding [5]. In terms of structure and development they should, again like the human cortex, be *modular* and *constructivist* [17]. In terms of efficiency, they should, like humans and animals, be able to do *one-shot learning*, i.e., learning from one or very few instances. For efficiency purposes, the algorithm should also be parallelisable. The work presented in this paper is a first step toward bringing these properties together in that it explores the principles of hidden state identification and discounted future reward estimation in a hierarchical, one-shot learning algorithm with the view to accommodating auto-associativity, modularity and constructivism in the future.

Section 2 of this paper places the work in the wider RL context. Section 3 presents the new hierarchical memory structure in detail along with the corresponding mechanisms for discounted future reward estimation and hidden state identification. Section 4 presents an experimental analysis of how the length of the sub-sequences affects the degree of re-use achieved as well as comparative experimental evidence indicating that the speed of the underlying NSM learning algorithm is not affected by the hierarchical memory representation. Section 5 presents an experimental analysis of the persistence of the shared sub-sequences. Finally, Section 6 concludes and indicates how we aim to extend this work in the future.

2 Related Work

Learning task hierarchies was identified by Dietterich [3] as an important challenge in RL and this was re-emphasised by Barto and Mahadevan [1]. Multiple frameworks for hierarchical RL have already been published, the most influential being the *options* formalism introduced by Sutton *et al.* [21]. Barto and Mahadevan [1] have presented a review of these and related frameworks. A lot of work has already been done on developing algorithms for hierarchical RL that automatically construct the abstractions used for the hierarchical representations. Digney [4] has presented the *Nested Q-learning* algorithm for this problem. Nested Q-learning uses environmental *features*, distinct recognisable sensory conditions to partition problem spaces. McGovern [15] has presented multiple algorithms for automated discovery of *sub-goals* based on bottle-necks defined by observation density in successful trajectories and *action sequences* based on frequency in successful trajectories. Hengst [6] has presented the *HEXQ* algorithm which uses the frequency of change in the state variables to define levels in the hierarchy. Sun and Sessions [19] have shown that, based on reward only, it is possible to construct dynamically and without prior knowledge of the problem domain, a hierarchy of individual agents who divide any given problem space in to beneficial sub problems and learn to solve these more efficiently than flat solutions. Common to these algorithms is their reliance on a statistical analysis of the input, something that requires a large number of trials in order to identify exploitable properties of the data. As such these algorithms are not *one-shot*.

Moore and Atkeson [16] have demonstrated that instance-based RL algorithms can improve on the speed of traditional algorithms such as temporal difference (TD) learning [20]. Other work in this direction deals with one-shot learning in terms of memory-based learning, e.g., locally weighted regression for robot juggling [18] and nonlinear oscillators for learning tennis strokes through programming by demonstration [8]. While these algorithms are one-shot, they are not hierarchical, and as such are unlikely to scale up to more general, less constrained problems.

Finally, at least two algorithms have integrated hierarchical representations with one-shot learning. McCallum [13] encoded trace instances hierarchically in the Utile Suffix Memory and U-Tree algorithms. These algorithms perform a much more efficient compression of memory than our algorithm in that it only distinguishes between states when there is a significant effect on the expected reward. However, there is no obvious way of implementing these algorithms using auto-associative structures and as such, these algorithms have little potential for further compression. Hernandez-Gardiol and Mahadevan [7] showed that hierarchical instance-based frameworks can improve the speed of learning beyond what is currently being achieved with flat representations. Hernandez-Gardiol and Mahadevan’s work however, looked in particular at representing problems at different levels of granularity in pre-structured hierarchies. As a result, the algorithms are not as generally applicable as those that automatically construct hierarchical representations.

Our work approaches the problem of task hierarchies from the angle of reuse for compression and uses syntactically defined hierarchies primarily to reduce the memory requirements of NSM. The reduction in memory requirement is achieved by constructing, dynamically and without prior domain knowledge, a LTM where finite sequences of transitions are shared between traces. Reduced memory requirements is an important factor in scaling up RL algorithms to handle increasingly complex problems. In Section 6 we discuss how this work is the first step in our work on developing RL algorithms that are both constructivist and auto-associative.

3 NSM with Hierarchical Traces

Kaelbling *et al.* [9] have presented an RL formalism for problems with hidden state, i.e., partially observable Markov decision problems (POMDPs). Basic Markov decision problems (MDPs) are described as a four-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where \mathcal{S} is a finite set of states of the world, \mathcal{A} is a finite set of actions, \mathcal{T} is the state transition function and \mathcal{R} is the reward function. To describe POMDPs Kaelbling *et al.* extend the MDP framework to a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ where Ω is a finite set of observations an agent can make of its world and \mathcal{O} is the state observation function giving the probability of making a particular observation given a specific state and action. An OARO *transition* is a quadruple $\langle o, a, r, o' \rangle$ where o is a starting observation, a is an action, r is the reward received and o' is the observation made in state s' resulting from taking action, a in the initial state s .

The original NSM learning algorithm was developed by McCallum [14] to be the simplest conceivable instance based RL algorithm. NSM learning is based on the k -nearest neighbours principle. It keeps track of the n latest observed OARO transitions in STM. In LTM it keeps the m most recently observed traces, where n and m are both fixed numbers. From the traces, NSM identifies the k OARO transitions that match most closely the last transitions stored in STM. The proximity measure used by NSM is the number of immediately preceding transitions in LTM that exactly match the immediately preceding transitions in STM. To select an action, NSM calculates the average discounted future reward (DFR) for each possible action over the k nearest neighbours. The NSM algorithm, in spite of being very simple, has been shown to compare favourably with other RL algorithms for problems with hidden state.

3.1 Hierarchical Traces

Figure 1 presents two different paths to a goal location in a grid-world originally presented by Sutton *et al.* [21].

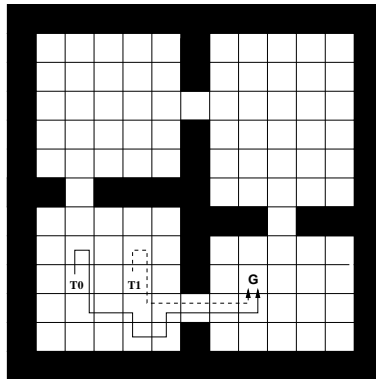


Fig. 1 Two different but overlapping paths to a goal location in the grid-world originally presented by Sutton *et al.* [21].

The values of the OARO transitions corresponding to the traces in Figure 1 are given in Table 1. Each row in Table 1 presents the values of a transition record as well as the corresponding DFR. We denote individual observations using the decimal representation of a four bits number where each bit indicates whether it's possible or not to enter the grid location to the north, east, south and west of the agent, respectively, starting from the least significant bit, e.g., a south-west corner with walls to the south and west would be represented by the bit string 1100 and the decimal value 12. The available actions are movements in the same four directions,

Hierarchical Traces for Reduced NSM Memory Requirements

ii) in the same order, represented as integers between 0 and 3 respectively. The reward function used yields a value of -1.0 when an agent attempts to move into a location which is occupied, a value of -0.1 when moving to an adjacent unoccupied location and a value of 5.0 when reaching the goal state. The DFR is calculated using a discount factor, γ , of 0.9 .

Trace 0					Trace 1						
Record	o	a	r	o'	DFR	Record	o	a	r	o'	DFR
0	0	0	-0.1	0	0.88	12	0	0	-0.1	0	1.87
1	0	2	-0.1	0	1.09	13	0	2	-0.1	0	2.19
2	0	2	-0.1	0	1.32	14	0	2	-0.1	0	2.54
3	0	1	-0.1	0	1.58	15	0	1	-0.1	0	2.95
4	0	1	-0.1	0	1.87	8	0	1	-0.1	5	3.37
5	0	2	-0.1	4	2.19	9	5	1	-0.1	0	3.86
6	4	1	-0.1	6	2.54	10	0	1	-0.1	0	4.4
7	6	0	-0.1	0	2.95	11	0	0	5.0	0	5.0
8	0	1	-0.1	5	3.37						
9	5	1	-0.1	0	3.86						
10	0	1	-0.1	0	4.4						
11	0	0	5.0	0	5.0						

Table 1 The transition records of the two traces presented in Figure 1, including the discounted future rewards.

Each level of our hierarchical trace representation contains a set of fixed length sequences. The number of levels is only restricted by the size of STM. On the lowest level of the hierarchy, the elements in the sequences are references to single OARO transition records. On all the levels above, the sequence elements are references to sequences on the level below. For sequence size n , a sequence on level 0 will contain references to n transition records while a sequence on level m will contain references to n^{m+1} transition records. The hierarchical trace representation makes it possible for two traces with common sub-sequences to share a single record of that sub-sequence. Individual OARO transition records are also shared by multiple traces and common sub-sequences within a single trace can also share a single record.

The hierarchical trace representations are constructed whenever a goal location is reached. The hierarchical NSM algorithm (HNSM) then structures the trace recorded in STM into a hierarchy. Each of the sequences in this hierarchy is then compared to the sequences recorded in LTM and, if two sequences are identical, any references to the new copy in STM is replaced by references to the existing copy in LTM. Any new transition records or sequences are added at the appropriate level. Remaining transition records not included in the fixed size sequences are recorded in shorter sequences. When removing a trace from LTM, the component transition records and sequences are only removed if they are not referenced by other traces. The hierarchical representation of the traces representing the two paths in Figure 1 are presented graphically in Figure 2. The boxed numbers are references to the sequences or transition records on the level below. On level 0 they reference the OARO

transition records presented in Table 1. Note that both traces make use of sequence 2 on level 0. This is reflected in Table 1 by both traces having the same four last records.

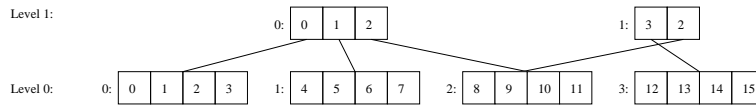


Fig. 2 A graphical representation of the traces representing the two paths presented in Figure 1 with sequence 2 on level 0 being shared by both traces.

3.2 Estimating Discounted Rewards

Calculating the DFR for each transition in a given trace is trivial with a linear representation. Here we present a mechanism for calculating DFR values using our hierarchical trace representation. In the hierarchical representation, all *elements* of a trace, i.e., both individual transition records and transition sequences, can be reused. It follows that each element can have multiple DFR values reflecting the different *contexts* in which they occur. We define a *local context* as a unique occurrence of an element in a superseding sequence. Note that an element can occur in multiple places in the same superseding sequence and also in different superseding sequences. Local contexts are indicated graphically by the lines connecting sequences and records across layers Figure 2. We define a *global context* as a unique occurrence of an element within a given trace.

Recording discounted rewards in the transition records means that two transition sequences are considered different if they occur at different distances from the final reward. This reduces the amount of calculation required at the cost of a lower level of memory reuse. We further reduced the memory requirements of our algorithm by recalculating the discounted rewards every time an action is chosen. We call this procedure *discounting rewards dynamically* (DRD). As DRD reliably reproduces the discounted rewards for all transitions in all the traces in LTM, we don't need to record rewards in the transition records. A given element can now be reused at any point in any trace.

An element that is used in multiple contexts correspondingly has multiple DFR values. The *local DFR* of a transition record or a sequence is the discounted sum of rewards recorded by that element. We denote the local DFR of memory element x in local context c , $d(x, c)$. The global DFR of a memory elements is the discounted sum of all the rewards recorded by preceding transition records in the given trace. In order to calculate the global DFR values of an element, the global DFRs of its superseding elements must be discounted over the number of transition records referred to by the elements preceding x in the given local context c . We denote this

distance $f(x, c)$. The discounted local DFRs for the preceding elements must then be added to the discounted global DFR of the superseding element. A recursive DFR definition is formalised in Equation 1. We use $D(x)$ to denote the set of global DFRs of an element and d' to denote a single global DFR for a superseding element. An element's set of local contexts is denoted $C(x)$. An element with no contexts has no superseding elements and thus, represent a complete trace.

$$D(x) = \begin{cases} \{0\} & \text{if } C(x) = \emptyset \\ \{d(x, c) + \gamma^{f(x, c)} d' \mid c \in C(x), d' \in D(c)\} & \text{otherwise} \end{cases} \quad (1)$$

Konidaris and Hayes [11] have provided evidence that DRD need not be computationally crippling. They have presented a mechanism called *asynchronous* RL within a greater behaviour-based RL framework. Asynchronous RL is a Q-learning mechanism that does *full backups* after each observed transition. A full backup improves a policy by iteratively refining it using estimated values to replace observations. This potentially requires revising the complete policy many times. Unlike asynchronous RL, DRD reproduces the same discounted rewards at each step and does not incrementally refine its estimates. What is important about Konidaris and Hayes' work in this context is their defence of the computationally expensive procedures of full backups on the basis that "...the time it takes a situated agent to move in the real world is very much longer than required to perform an update..." This argument also applies to DRD and Konidaris and Hayes' experimental evidence supports the feasibility of such approaches in general.

3.3 Hidden State Identification

In the same way that we developed a new mechanism for calculating DFR values for hierarchically represented traces, we have developed a corresponding mechanism for calculating the proximity values used to identify the k -nearest neighbours for handling hidden states. A *local proximity* is the number of matches between the transition records in STM and the transitions referred to by the preceding elements in a local context in LTM. A *global proximity* is the number of matches between STM and the transitions referred to by a global LTM context. However, only when STM matches all the preceding elements of a local context, is it necessary to check for further matches in higher level contexts. If there are further matches, the local proximity must be added to all of these to find the global proximities. A recursive definition of global proximity is formalised in Equation 2. The local proximity of an element given a context c is denoted $p(x, c)$ and the set of global proximities of an element $P(x)$. We also use $P(c)$ to denote the global proximity of a context, c . We use p' to denote the global proximity for a superseding element. Note that the global proximities are only added when STM matches the local context completely.

$$P(x) = \begin{cases} \{0\} & \text{if } C(x) = \emptyset \\ \{p(x,c) \mid c \in C(x)\} & \text{if } c \text{ matches STM incompletely} \\ \{p(x,c) + p' \mid c \in C(x), p' \in P(c)\} & \text{if } c \text{ matches STM completely} \end{cases} \quad (2)$$

We have used this definition of proximity to implement a function which calculates all possible proximities for each transition record. From these global proximities we choose the k highest and execute the action with the highest average DFR.

4 Re-use and Sub-Sequence Length

Our algorithm currently uses fixed-length sub-sequences, i.e., is syntactically defines the hierarchical structure of the LTM. However, the hierarchical hidden state identification and DRD mechanisms also support variable length sub-sequences. This makes it possible to vary the sub-sequence length to optimise memory compression or to consider semantic rather than syntactic criteria for sub-sequence lengths, e.g., the amount of re-use can be used to allow heavily referenced sub-sequences to grow longer, or the strength of reward signal can be used to allow sub-sequences with high DFR values to be longer than sub-sequences with low DFR values. Such mechanisms provide a rich area for exploring and optimising memory usage. We discuss these opportunities further in Section 6. For fixed size sub-sequences, the optimal sub-trace size is not obvious and is likely to be problem specific. However, a sub-trace size of 1 would not provide any scope for compression, so the smallest possible sub-trace size is 2. Increasing the sub-trace size reduces the chance of matching sub-sequences. A sub-sequence size larger than the size of STM would mean that only complete traces would be shared. We have analysed the memory requirements by counting the number of values and references stored when using sub-sequences of size 2, 4 and 8 on two different problems described below. We have also analysed the memory usage for the original NSM algorithm.

For simplicity we use an abstract memory cost of 1 for observations, actions, resulting observations, rewards and DFR values as one each. A reference from a sequence to a sub-element or super-sequence also has a cost of 1. Each transition record in the original NSM algorithm thus has a memory requirement of 5 and a trace containing n transition records has a memory requirement of $5n$. Our algorithm dynamically discounts the rewards, thus the memory requirements of each transition record is 4 as we do not store the DFR value. However, in addition to the transition records, our algorithm needs two-way references between elements on different levels in the hierarchy in order to support hierarchical hidden state identification and DRD. While each transition records has a memory requirement of $4n$, there is also the overhead of hierarchical references. A complete hierarchical trace without reuse contains $2(n \log n)$ references and has a total memory requirement of $4n + 2(n \log n)$. The actual number of transition records and references used by our algorithm will be smaller than this as different traces will share sub-sequences. We

ran our algorithm in two different abstract mazes, or *grid worlds*¹, taken from the RL literature. The first grid-world was originally published by Sutton [21] and is presented graphically in Figure 1. The second was originally published by by McCallum [14] and is presented graphically in Figure 3.

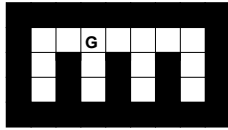


Fig. 3 The simplest of the two grid worlds used for the experiments, originally presented by McCallum [14].

All the experiments used the reward function and discount factor given in Section 3.1. The number of nearest neighbours, k , was set to 7, and the exploration rate was fixed at 0.1. The size of the STM, i.e., the maximum trace length, was 40 and the size of the LTM, i.e., the maximum number of traces stored, was also 40. Each experiment consisted of 40 trials and each trial consisted of 200 runs. The average memory usage for McCallum’s world is presented graphically in Figure 4. The usage for Sutton’s world is presented in Figure 5. The different versions of our algorithm, using DRD and hierarchical traces is labelled $DHNSMn$, where n denotes the sub-sequence length. The average memory usage for the original NSM algorithm is also given. The x-axis gives the number of runs the algorithms have gone through while the y-axis gives the average memory usage.

The results presented in Figures 4 and 5 show that in both grid-worlds, the lowest memory requirements are achieved when using a sub-trace size of 4. We also performed experiments using sub-trace sizes of 3, 5, 6 and 16. Sub-sequences of 3 and 5 produce a memory usage that is not significantly different from those produced by sub-sequences of size 4. Sub-sequences of size 6 and 16, like 8, produce a memory usage that is significantly higher. For the experiments on learning speed and sub-sequence persistence below, we only consider sub-sequences of size 4.

4.1 Speed of Learning

The original NSM algorithm and our algorithm are effectively the same algorithm with different memory requirements. As a point of interest, when debugging our algorithm, we ran the two in parallel and compared the proximities, DFRs and actions selected in order to ensure that they were exactly the same. As a result, the perfor-

¹ All our experiments were conducted using the Rumpus open source, stand-alone grid world server which supports easy development of new RL problems using images and XML as well as multi-language support through TCP sockets. The Rumpus server along with all its implementations of RL problems and algorithms are available under the GNU general licence from <http://rumpus.rubyforge.org>.

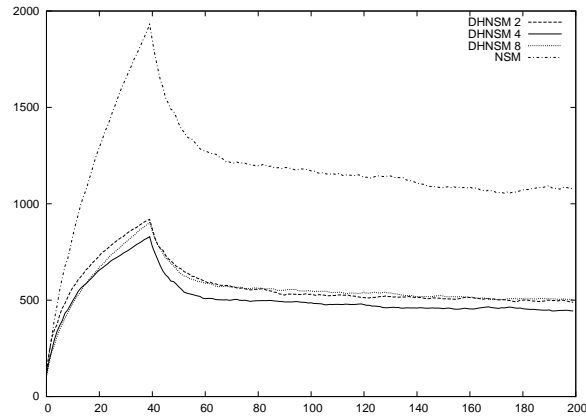


Fig. 4 The memory usage of our DHNSM algorithm using different sub-sequence lengths in McCallum's grid world.

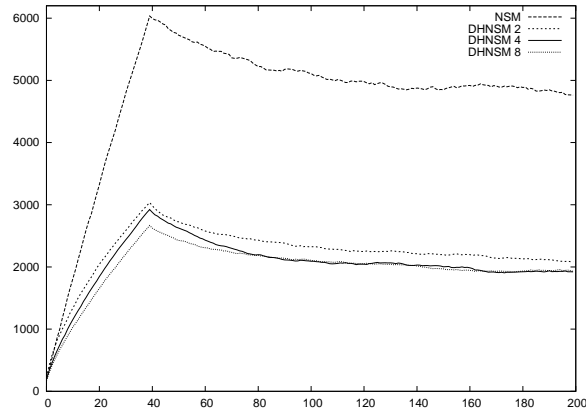


Fig. 5 The memory usage of our DHNSM algorithm using different sub-sequence lengths in Sutton's world

mance in time should be indistinguishable. In Figure 6 we present graphically the performance in time from the experiments in McCallum's grid-world described above. In Figure 7 we present the same data for Sutton's world. Again the x-axis gives the number of runs the algorithms have gone through while the y-axis gives the average number of steps needed to reach the goal.

The data presented in Figures 7 and 6 indicate that there is no significant difference in terms of the speed of learning, between the original NSM algorithm and our algorithm independent of sub-sequence length.

Hierarchical Traces for Reduced NSM Memory Requirements

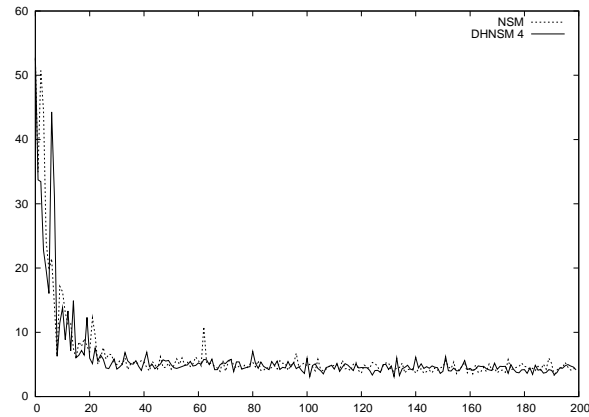


Fig. 6 The average number of steps needed to complete each run for the DHNSM 4 and NSM algorithms in McCallum's world

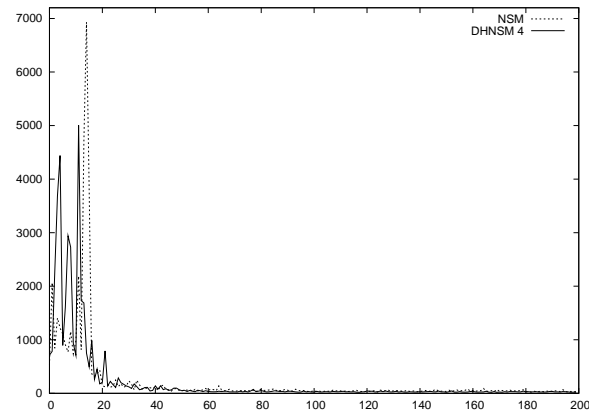


Fig. 7 The average number of steps needed to complete each run for the DHNSM 4 and NSM algorithms in Sutton's world

5 Memory Persistence

As hierarchical traces are made up of shareable sub-sequences, there is a potential for sub-sequences to out-live the traces that originally created them. Sub-sequences that persist beyond the lifetime of any individual trace describe commonly recurring agent-world interaction sequences. Sub-sequences that persist throughout the learning process describe interactions that are both common and beneficial in that they are part of the successful traces that are repeated more frequently as the agents learn. In Figure 8 we present the maximum, average and minimum age of the sub-

sequences from McCallum’s world. Figure 9 presents the same data from Sutton’s world.

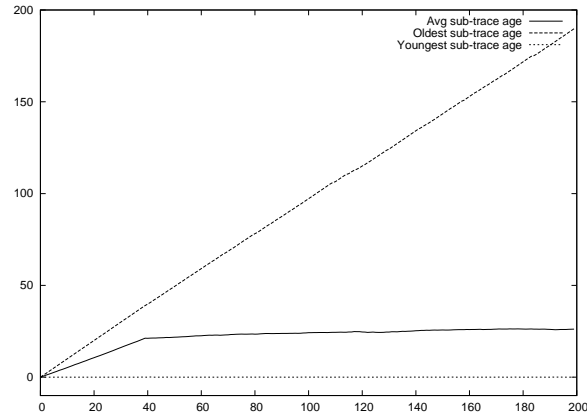


Fig. 8 The maximum, average and minimum age of the sub-sequences in LTM for the DHNSM 4 algorithm in McCallum’s world

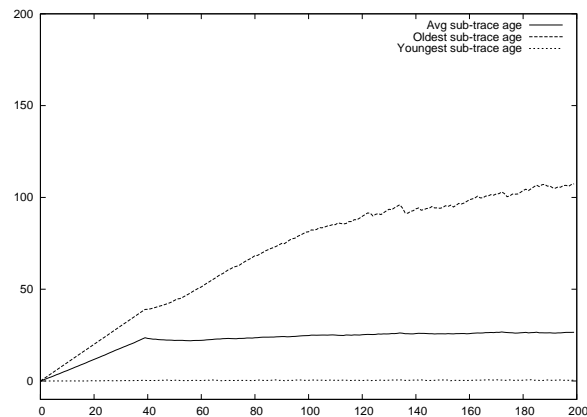


Fig. 9 The maximum, average and minimum age of the sub-sequences in LTM for the DHNSM 4 algorithm in Sutton’s world

Figures 9 and 8 both show the presence of a persistent core of sub-sequences. If sub-sequences did not outlive their creators, the average sub-trace length would be 20. For both worlds, the average grows to over 25 with the maximum age growing to over 100. These persistent elements describe the agent-world interaction by representing its most common transition sequences. As such these elements form a bridge between instance-based and model-based learning algorithms. The presence

of persistent elements of sub-sequences is also a promising sign for our future development of this algorithm in a more obviously model-based direction as described in Section 6.

6 Conclusions and Future Work

The results presented in this paper show that it is possible to use hierarchical traces for instance-based RL and that such traces can reduce the memory requirements of instance-based RL algorithms. While variable sub-sequence sizes are an interesting direction in which to take this research, another direction of enquiry provides a more compelling memory compression mechanism. We plan to implement sub-sequences as self-organising maps (SOMs) [10] and by doing this, we hope to achieve three things. First, we hope to provide further memory compression by representing similar traces through their principal components. Second, we hope to provide RL algorithms that are at once, sequential, hierarchical and auto-associative. Third, by implementing DFR and hidden state identification in terms of *activation spreading* [12] in a hierarchical SOM, we aim to produce RL algorithms that are massively parallelisable on a low, non-von Neumann architectures such as field programmable gate arrays. Chang *et al.* have already demonstrated such parallelisation of traditional SOMs [2].

Such algorithms will have increased scalability in space and time. The work presented here gives us a formal yard-stick for evaluating the performance of such a mechanism.

References

1. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications* **13**, 41–77 (2003)
2. Chang, C.H., Shibu, M., Xiao, R.: Self organizing feature map for color quantization on fpga. In: A.R. Omondi, J.C. Rajapakse (eds.) *FPGA Implementations of Neural Networks*. Springer (2006)
3. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* **13**, 227–303 (2000)
4. Digney, B.L.: Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement learning. In: P. Maes, M. Matarić, J.A. Meyer, J. Pollack, S.W. Wilson (eds.) *From Animals to Animats 4* [Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB'06), Cape Cod, Massachusetts, September 9 - 13, 1996], pp. 363–372. MIT Press/Bradford Books (1996)
5. Fuster, J.M.: *Cortex and Mind: Unifying Cognition*. Oxford University Press, New York (2003)
6. Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In: C. Sammut, A.G. Hoffmann (eds.) *Machine Learning* [Proceedings of the 19th International Conference (ICML'02), Sydney, Australia, July 8 - 12, 2002], pp. 243–250. Morgan Kaufmann (2002)
7. Hernandez-Gardiol, N., Mahadevan, S.: Hierarchical memory-based reinforcement learning. In: T.K. Leen, T.G. Dietterich, V. Tresp (eds.) *Advances in Neural Information Processing Sys-*

- tems 13 [Proceedings of the NIPS Conference, Denver, Colorado, November 28 - 30, 2000], pp. 1047–1053. MIT Press (2001)
8. Ijspeert, A.J., Nakanishi, J., Schaal, S.: Movement imitation with nonlinear dynamical systems in humanoid robots. In: Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA'02), pp. 1398–1403. Washington, DC (2002)
 9. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. *Artificial Intelligence* **101** (1998)
 10. Kohonen, T.: Self-organizing maps, 3rd edn. Springer, New York (2001)
 11. Konidaris, G.D., Hayes, G.M.: An architecture for behavior-based reinforcement learning. *Adaptive Behavior* **13**(1), 5–32 (2005)
 12. Maes, P.: How to do the right thing. *Connection Science* **1**(3), 291–232 (1989)
 13. McCallum, A.: Instance-based utility distinctions for reinforcement learning with hidden state. In: Proceedings of the 12th International Conference on Machine Learning (ICML'95), pp. 387–395. Tahoe City, California (1995)
 14. McCallum, A.: Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics (Special Issue on Robot Learning)* **26**(3), 464–473 (1996)
 15. McGovern, A.: Autonomous discovery of temporal abstractions from interactions with an environment. Ph.D. thesis, University of Massachusetts, Amherst, Amherst, Massachusetts (2002)
 16. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* **13**, 103–130 (1993)
 17. Quartz, S.R.: Learning and brain development: A neural constructivist perspective. In: P.T. Quinlan (ed.) *Connectionist Models of Development: Developmental Processes in Real and Artificial Neural Networks*, pp. 279–310. Psychology Press (2003)
 18. Schaal, S., Atkeson, C.G.: Robot juggling: Implementation of memory-based learning. *Control Systems Magazine* **14**(1), 57–71 (1994)
 19. Sun, R., Sessions, C.: Self-segmentation of sequences: Automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems, Man and Cybernetics: Part B, Cybernetics* **30**(3), 403–418 (2000)
 20. Sutton, R.S., Barto, A.G.: *Reinforcement learning: an introduction*. MIT Press, Cambridge, Massachusetts (1998)
 21. Sutton, R.S., Precup, D., Singh, S.P.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* **112**, 181–211 (1999)