Analysis of Memory-Based Learning Schemes for Robot Navigation in Discrete Grid-Worlds with Partial Observability

Bharadwaj Srinivasan
SRM Engineering College, Anna University
No.18/2, Velu Street, West Mambalam, Chennai – 600033, India
91-44-24748364
bharadwaj14@gmail.com

Abstract

In this paper we tackle the problem of robot navigation in discrete grid-worlds using memory-based learning schemes. Different memory-based approaches are tested for navigating an agent across a discrete but partially observable world, and the significance of memory structure is examined. Further, the effects of additional memory hierarchies and multi-level learning frameworks are analyzed with regard to two contrasting Reinforcement Learning algorithms. The learning problem is formulated using a POMDP model and the agent makes use of Temporal Difference learning methods to learn an optimal navigation policy with the help of a short-term memory.

Keywords

Robot Navigation, Memory-Based Learning Systems, Q-Learning, SARSA

Introduction

In this paper, we attempt to find a solution to the problem of robot navigation using different memory-based learning schemes, and compare these schemes in terms of learning speed and efficiency.

The problem of robot navigation dealt with in this paper, is essentially that of finding an optimal navigable path for the robot in a given environment, with certain constraints imposed on the robot, such as a time limit or limited availability of resources. *Optimal path* here refers to a path between the two points say the source and destination, which has the least path cost, or in other words the most profitable one among all the existing paths.

The environment is a discrete grid-world partitioned into rooms and hallways, with randomly located obstacles. The agent, which is nothing but a simple mobile robot, can occupy a single non-empty tile at a time and is faced with the task of navigating through the map in an autonomous manner. The agent is capable of sensing its immediate environment and moving in any direction, one tile at a time.

The most significant feature of this problem is the concept of partial observability. Partial observability implies that the agent is at no point fully aware of its surroundings, and can only perceive a part of it at any given time. This complicates the problem of navigation since there is always a level of uncertainty about the agent's position, relative to the environment. For example, the agent may sense the same perceptions at different points on the map, thereby rendering it unable to distinguish between those points.

The approach used in this paper involves an autonomous learning technique, namely Reinforcement Learning. The agent is programmed to 'learn' how to navigate the map, by autonomously exploring different areas of the map repeatedly and acquiring knowledge from its experience. We supply the agent with a short-term memory which it uses to store past information and learns to take optimal actions based on its past experiences.

Further, we test different Temporal Difference learning algorithms [11] on the agent, with and without the use of short-term memory and also vary the structure and organization of memory as well as the learning scheme, and finally present a comparative analysis of these techniques.

Problem description

The problem statement is simple terms is to develop an efficient learning system for the agent to autonomously navigate through any given grid-world environment.

When modeled as a Reinforcement Learning System, the agent is provided with a reward for reaching the destination, and is penalized for colliding into walls or obstacles. Also, each step taken by the agent has an associated step cost, that the agent must aim to minimize.

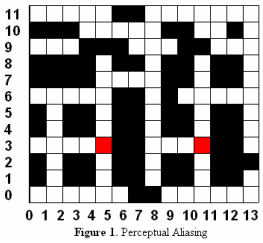

Figure 1. Perceptual Aliasing

A sample grid-world scenario is shown in Figure 1. Here, a major problem faced by the agent is that the state space can be very large, making it difficult for the agent to keep track of the world as a whole. Further, partial observability leads to the problem of perceptual aliasing, ie. Different real world states generate the same observation to the agent. For example in the adjacent map, both the tiles (4,3) and (10,3) marked in red are indistinguishable to the agent.

In order to overcome these hurdles, we adopt a memory-based approach, described in the later sections of this paper.

Problem formulation

The entire state space and actions taken by the agent are formalized as a finite *Markov Decision Process* (MDP)[8] A finite Markov Decision Process is a tuple <S, A, Ψ, P, R>, where *S* = {1, 2, 3, …, n} is a set of states, *A* is a finite set of actions, Ψ S x A, is the set of admissible state-action pairs, P : Ψ x S → [0,1], is the transition probability function with P(*s,a,s'*) being the probability of transition from state *s* to state *s'* under action *a*, and R : Ψ → **R** is the expected reward function, with R(*s, a*) being the expected reward for performing action a in state *s*.

However, due to partial observability criteria, the agent does not perceive the entire state space *S*, but only a set of observations, say *O*. Thus the formalization is actually that of a *Partially Observable MDP* (POMDP) [9], which is a generalization of an MDP, with *O* denoting the set of all observations perceived by the agent, and rest of the parameters remaining the same.

The agent thus tries to learn an optimal *Control Policy* [11] which is nothing but a relation C : O → A, giving the optimal action to be chosen by the agent, corresponding to each observation.

Solution

An autonomous learning system has been implemented using two alternative Temporal Difference learning algorithms, namely Q-learning [8] and SARSA [11].

Temporal Difference Learning

As in any Reinforcement Learning scheme, the aim here is to maximize the cumulative reward obtained by the agent. As mentioned earlier, the agent is presented a certain reward when it reaches the destination and a certain penalty for colliding with walls and obstacles.

A certain *Reward Function* can thus be associated with every State-Action pair (*s,a*) which is nothing but the reward *r* obtained when the agent chooses the action *a* in state *s*. Thus the agent's goal is to maximize the total reward obtained over time, or in other words maximize the final *return.*

Each state-action pair is associated with a value function (Q-function) which gives the desirability of choosing an action in that state. Thus Q(*s,a*) gives the desirability of choosing action *a* in state *s* and is initialized with random values for all *a*. The Q-functions for each state-action pair are updated as the learning progresses, and in Temporal Difference (TD) learning, this update takes place based on the Q-value of the successive state-action pair.

An Є-greedy algorithm [11] is used to pick the action at each step, thereby choosing a random action with probability Є, and the action with highest Q-value, with probability 1- Є, where Є is a small value corresponding to the exploration factor. This process of selection and update is repeated until the goal state is reached and the learning stops. Training takes place by repeating the above process over numerous iterations of learning until the Q-values stabilize to the desired level of accuracy.

SARSA Algorithm

Initialize Q(s,a) arbitrarily
Repeat (for each episode)
  Initialize s
  Choose a *from* s *using policy derived from Q (ie. Є-Greedy)*
  Repeat (for each step of episode)
    Take action a*, observe* r*,* s' *(where* s' *is the new state reached)*
    Choose a' *from* s' *using policy derived from Q (ie. Є - Greedy)*
    $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma.Q(s',a') - Q(s,a)]$
    s ← s' *;* a ← a'
  until s *is terminal*

Figure 2. SARSA Algorithm

Q-Learning Algorithm

Initialize Q(s,a) arbitrarily
Repeat (for each episode)
  Initialize s
  Repeat (for each step of episode)
    Choose a *from* s *using policy derived from Q (ie. Є - Greedy)*
    Take action a*, observe* r*,* s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma. \max_{a'} Q(s',a') - Q(s,a)]$
    s ← s' *;*
  until s *is terminal*

Figure 3. Q-Learning Algorithm

SARSA is an on-policy TD algorithm in the sense that the updates to Q-values are made at each step based on the next selected action, and not the best possible action at the next step, as is done in Q-Learning, which is an off-policy TD algorithm.

In our case, since the set of all states S maps onto the set of all observations *O*, we replace state *s* by observation *o.*

Memory and Hierarchy

As discussed already, the two major hurdles faced in this problem are that of perceptual aliasing and scaling up to larger domains.

In order to overcome the problem of perceptual aliasing, we introduce the notion of a short-term memory. The agent will thus be able to remember a history of previously encountered states, and instead of taking decisions based on single states, it will consider a history of states as a whole. A history can hence be defined as a sequence of the last *n* observations and the agent's memory becomes a collection of all such histories that have been encountered in the past. Thus, a 'state' as perceived by the agent is redefined as a history of real-world states.

For the purpose of scalability to larger domains, we organize the learning process into different levels of a hierarchy, such that each level abstracts over minute details, which are present in the lower levels. Learning takes place at different hierarchical levels based on the histories of observations made at each level. This greatly speeds up the learning process and also simplifies the spatial complexity required for learning by a great deal.

Hence we go in for a hierarchy of memory levels, instead of a flat, one-level state space.

A technique of short-term memory that is used in this paper is known as *Nearest Sequence Memory* (NSM) [4] which records raw experiences in the form of a linear chain. Another form of memory representation known as the *Utile Suffix Memory* (USM) [4] has also been implemented. However, it is preferable to have the length of histories stored in the memory or in another words the memory length as a variable, in order to speed up the learning process and make it adaptable to the different forms of perceptual aliasing encountered in the map. For this purpose, a technique of variable memory, namely *U-Trees* [5] can be used. However, in this paper we restrict ourselves to memories of fixed length only.

The claims made in section 3.2 will be justified with adequate results in the analysis section.

Implementation

In this paper, we have implemented both the Q-Learning and SARSA algorithms to contrast the effectiveness of the two. Further, we have tested the agent in the following cases:

i. Without short-term memory
ii. With short-term memory (A comparison between different memory sizes has also been done)
iii. With a hierarchy of memory levels

The Hierarchy used in this particular solution consists of 2 levels that are defined as follows:

Level 1 (Higher Level): This level consists of landmarks on the map, which in this case are all corridor intersections. At this level, the agent picks a direction to move in.

Level 2 (Lower Level): This level consists of the hallways or corridors in the map and the agent learns to navigate through a corridor, avoiding obstacles. The actions available to the agent in the lower level are primitive actions such as Turn Left, Turn Right or Move Forward, or an *option* [12] consisting of a temporally extended combination of these actions.

Level 1 of the Hierarchy (Higher Level):

i. *Picks one of the available directions to move in, at an intersection*
ii. *Passes control to level 2 after exiting the intersection along that direction*

Level 2 of the Hierarchy (Lower Level):

i. *Uses an option to navigate within a corridor, avoiding obstacles*
ii. *The option is trained separately in a training corridor*
iii. *Terminates and passes control back to Level 1 on reaching an intersection*

Figure 4. Levels of the Hierarchy

Learning and Updating Q-functions:

In each of the two levels, the corresponding Q-function is updated after each move, using the SARSA Algorithm as follows:

Level 1:

$$Q(s,a) = Q(s,a) + \alpha\,[\,(\gamma^0 + \gamma^1 + \ldots + \gamma^{n-1}).r + \gamma^n.Q(s',a') - Q(s,a)\,] \quad (1)$$

where n = of steps taken from previous intersection, α = learning rate and γ = discount function

Level 2:

$$Q(s,a) = Q(s,a) + \alpha\,[\,r + \gamma.Q(s',a') - Q(s,a)\,] \quad (2)$$

The learning and updating process continues until the goal state is reached. As this is run over several iterations, the agent learns the optimal control policy C, which it then uses to navigate the map efficiently.

Testing and Analysis

The algorithms explained in this paper for navigating partially observable domains have been tested comprehensively across a wide range of maps, and the results obtained on 4 such maps are shown below.

Map 1:

- Size : 15 x 15
- Start : (4, 4)
- Destination : (12, 12)
- Optimal No. of Steps = 18

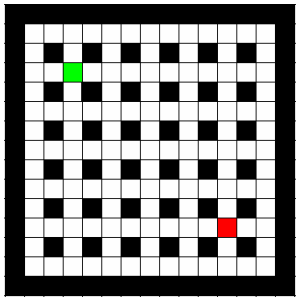

Figure 5. Map 1

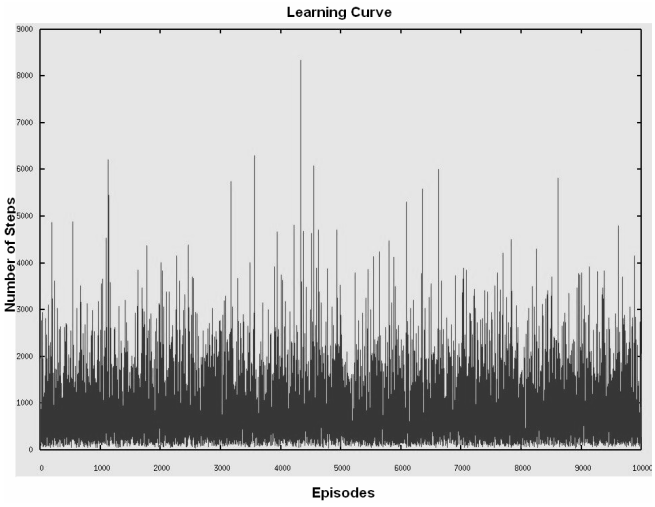

Figure 6. Map 1 – Learning Curve (No Memory)

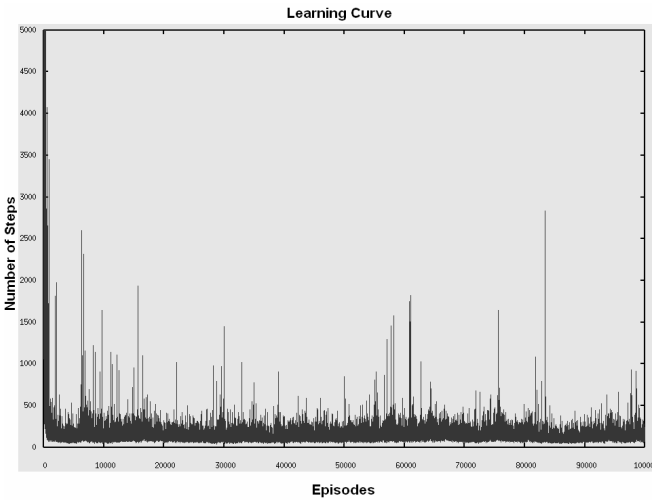

Figure 7. Map 1 – Learning Curve (Single Level Memory)

In (Figure 6) the curve is haphazard and the agent shows no signs of learning. This is due to the fact that the algorithm is unable to overcome the problem of perceptual aliasing without the use of memory.

In (Figure 7) however when one level of memory is added, the agent is able to learn the policy, as indicated by the learning curve. This highlights the importance of memory, especially in conditions where perceptual aliasing occurs.

Map 2:

- Size : 25 x 25
- Start : (6, 6)
- Destination : (20, 17)
- Optimal No. of Steps = 34

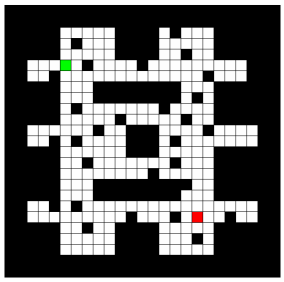

Figure 8. Map 2

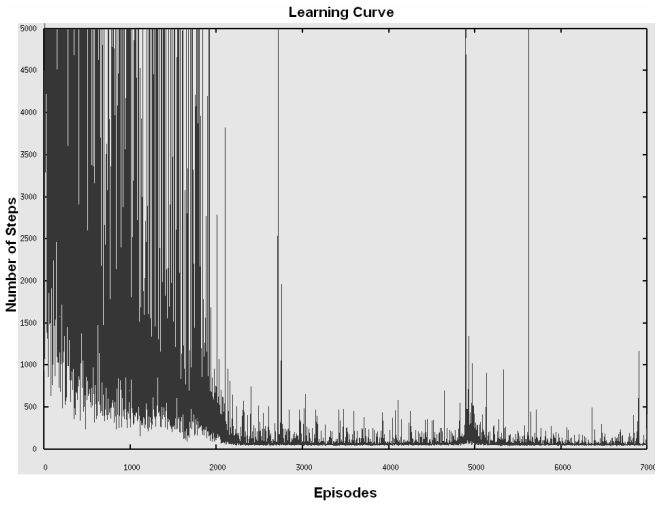

Figure 9. Map 2 – Learning Curve (Memory-size = 3)

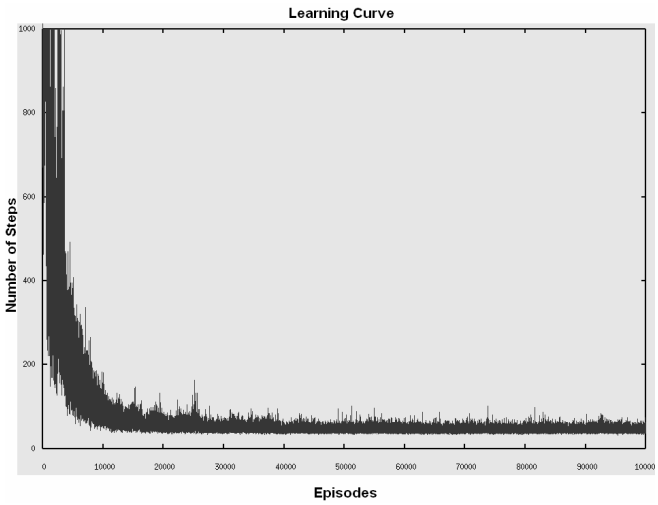

Figure 10. Map 2 – Learning Curve (Memory-size = 5)

(Figure 9) & (Figure 10) show the learning curves obtained when the agent uses a single level short-term memory of size = 3 states and 5 states respectively, in Map 2. It is evident from the two figures that the learning is more efficient in the case where memory-size = 5 states. Hence it is inferred that the size of memory used plays a very important role in determining the efficiency of learning, and thereby the navigation. Hence, as already mentioned in Section 3.2, the use of U-Trees that incorporate variable length memories is ideal for such cases.

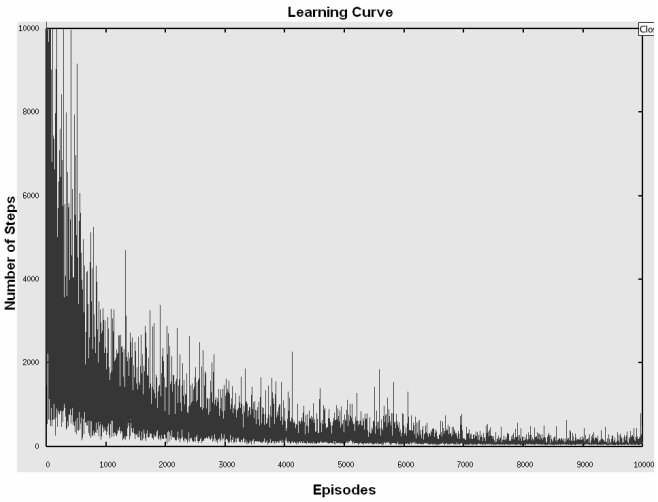

Figure 11. Map 2 – Learning Curve (Hierarchical Memory)

(Figure 11) shows the learning curve obtained when Hierarchical Memory is used by the agent, in Map 2. The curve resembles an ideal learning curve, indicating that the use of a two-level hierarchy greatly simplifies the learning task, as compared to a single level of memory.

Map 3:

- Size : 28 x 26
- Start : (7, 17)
- Destination : (20, 19)
- Optimal No. of Steps = 18

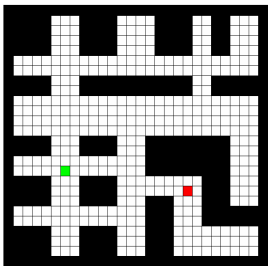

Figure 12. Map 3

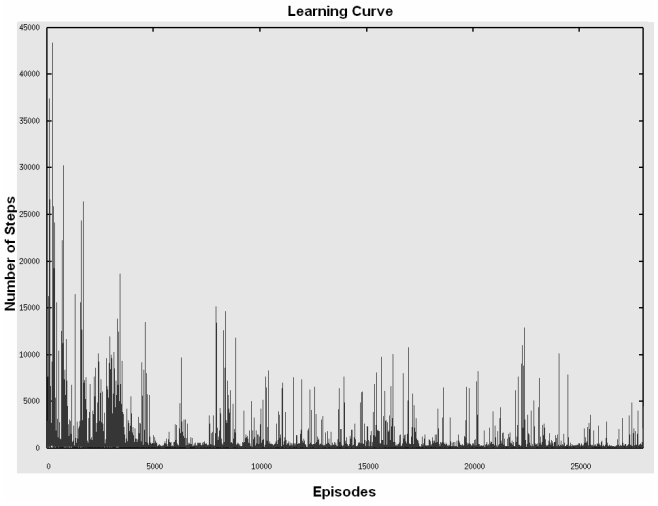

Figure 13. Map 3 – Learning Curve (Single Level Memory)

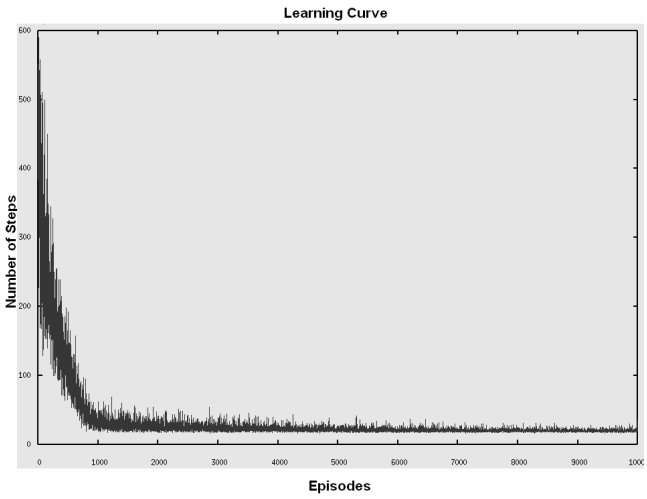

Figure 14. Map 3 – Learning Curve (Hierarchical Memory)

Again, the above two graphs establish the fact that Hierarchical Memory performs significantly better than a single level memory.

Map 4:

- Size : 12 x 11
- Start : (10, 3)
- Destination : (11, 9)
- Optimal No. of Steps = 38

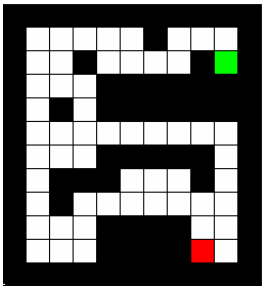

Figure 15. Map 4

We use Map 4 to compare Q-Learning and SARSA algorithms, both using a single level of memory.

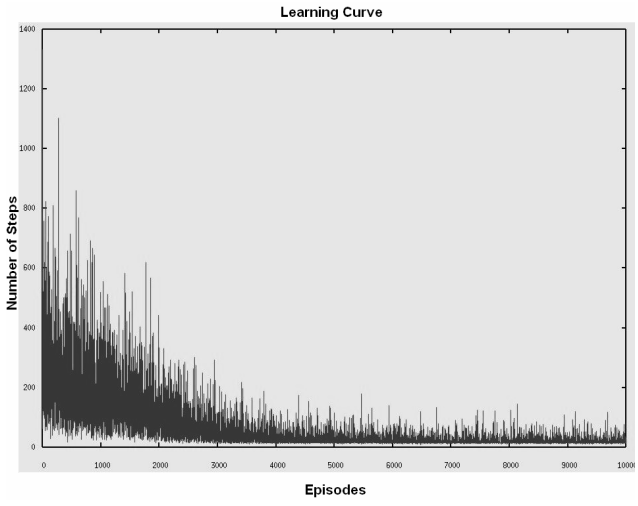

Figure 16. Map 4 – Q-Learning (Single Level Memory)

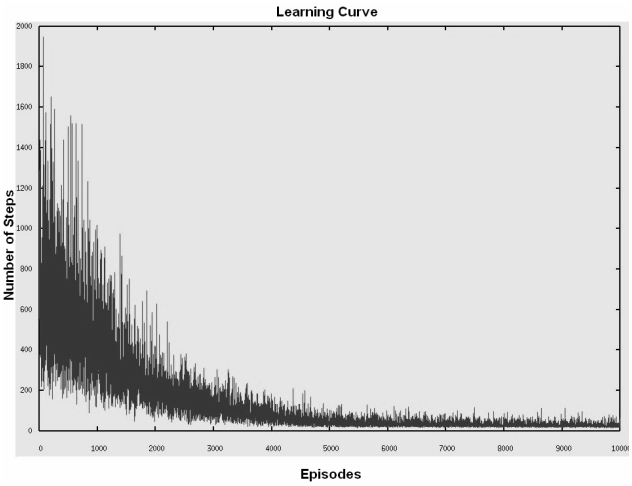

Figure 17. Map 4 – SARSA (Single Level Memory)

It is observed from (Figure 16) & (Figure 17) that though Q-Learning performs better than SARSA in the initial stages, it falls short of SARSA in the later stages and the learning is thus more stabilized in the case of SARSA algorithm. Thus, though Q-Learning learns actions according to the optimal policy, SARSA outperforms it unders such conditions. However as Є is gradually decayed, both curves will stabilize at the same level.

The agent is thus able to learn the optimal control policy most efficiently when it uses the Hierarchical Memory Based approach. But, we also notice that the learning process is significantly faster in the absence of obstacles on the map.

Future work

i. To switch to variable length memory using U-Trees.

ii. To implement goal regression and prioritized sweep.

iii. To add eligibility traces to speed up learning.

iv. Exploration vs Exploitation trade-off : To decay the value of Є gradually to obtain optimal solutions after learning has stabilized .

v. To incorporate transformations in histories to exploit symmetry and other similarities between histories.

Acknowledgements

I sincerely thank Dr.B.Ravindran, Assistant Professor, Department of Computer Science & Engg., IIT Madras, for his clear guidance and unfailing assistance, without which this paper would not have been possible.

I also thank the Head, faculty members, students and staff of the Department of Computer Science & Engg., IIT Madras, for their support and encouragement.

References

[1] Natalia Hernandez Gardiol and Sridhar Mahadevan, "Hierarchical Memory-based Reinforcement Learning", *Advances in Neural Information Processing Systems 13, (NIPS*2000). MIT Press, Cambridge*, 2001

[2] Bharadwaj Srinivasan, "Robot Navigation in Partially Observable Domains using Hierarchical Memory-Based Reinforcement Learning", *2nd International Conference on Ubiquitous Robots and Ambient Intelligence, 2005.*

[3] McCallum, R. Andrew, "Hidden State and Reinforcement Learning with Instance-Based State Identification", *IEEE Transations on Systems, Man and Cybernetics (Special issue on Robot Learning), to appear,* 1996.

[4] McCallum, Andrew K., Reinforcement "Learning with Selective Perception and Hidden State", *PhD. thesis. December*, 1995.

[5] McCallum, R. Andrew, "Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks", in *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior*, (SAB'96). Cape Cod, Massachusetts. September, 1996.

[6] Ravindran, B. and Barto, A. G. (2003) "An Algebraic Approach to Abstraction in Reinforcement Learning". In the *Proceedings of the Twelfth Yale Workshop on Adaptive and Learning Systems*, pp. 109-114. Yale University.

[7] S. D. Patek, "On Partially Observed Stochastic Shortest Path Problems," *Proceedings of the 40th IEEE Conference on Decision and Control (CDC 2001). IEEE Part vol. 5, pp. 5050-5055.*

[8] Parr, R. E. "Hierarchical control and learning for markov decision processes". *Doctoral Thesis*, 1998.

[9] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. "Learning hierarchical partially observable markov decision processs".

[10] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. "Learning hierarchical partially observable markov decision processs for robot navigation". *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[11] Richard.S.Sutton & Andrew.G.Barto, "Reinforcement Learning, An Introduction", *MIT Press*, 1998.

[12] Barto, A. G. and Mahadevan, S. "Recent Advances in Hierarchical Reinforcement Learning". *Discrete Event Dynamic Systems* 13, 4 (Oct. 2003), 341-379.