Robot Navigation in Partially Observable Domains using Hierarchical Memory-Based Reinforcement Learning

Bharadwaj Srinivasan
Department of Computer Science & Engg.
S.R.M. Engineering College,
Anna University, Chennai, India
E-mail: bharadwaj14@gmail.com

***Abstract* — In this paper, we attempt to find a solution to the problem of robot navigation in a domain with partial observability. The domain is a grid-world with intersecting corridors, where the agent learns an optimal policy for navigation by making use of a hierarchical memory-based learning algorithm. We define a hierarchy of levels over which the agent abstracts the learning process, as well as its behaviour. The problem is modeled as a POMDP and a solution is obtained by implementing the SARSA algorithm, which incorporates Temporal Difference learning. The agent uses short-term memory and abstracts over minute details thereby enabling it to scale up to large partially observable domains.**

Keywords — **Robot Navigation, POMDP, Hierarchical Reinforcement Learning, SARSA**

1. Introduction

The problem of robot navigation in general refers to that of finding a navigable path between any two points say, the source and the destination, on a given map. Assuming that at least one such path exists, the agent's task is to find the best path among all such existing paths. *Best* here refers to the most profitable path, or in other words, the path having the least overall cost. Thus such an optimal path must be found, with certain constraints being imposed on the agent, such as time limits and limited availability of resources.

In this paper, we tackle the problem of robot navigation in a domain with 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.

Path planning problems in domains with partial observability have been studied under AI for a long time, though their origins can be traced to problems rising in Operations Research. Of the various approached used to tackle planning problems in POMDPs, the most effective ones use a Belief State model [16] to approximate real-world states in a probabilistic manner. Other approaches include aggregation of states, maintaining a state history, and the use of a Predictive Representation of States [15].

The approach used in this paper, to guide the agent through the map is one of *Hierarchical Memory-Based Reinforcement Learning* [10]. The agent learns to take optimal actions at every step, by experimenting as it passes through the map, and is ultimately equipped with the knowledge of the optimal sequence of steps from source to destination.

The use of Hierarchical Memory-Based Reinforcement Learning for robot navigation has already been demonstrated [1]. The salient features of this paper, in comparison with the above mentioned work are as follows.

In this paper, the domain has been narrowed down to a discrete grid-world environment and the algorithm has been implemented on a generic gird-world platform created for this purpose. In [1] however, a continuous spatial domain serves as the environment and the Nomad 200 simulator [8] has been used for testing purposes. Further, this paper makes use of the on-policy SARSA learning algorithm, in contrast with the off-policy Q-learning approach [10] used in [1] and also incorporates an alternative short-memory technique. In addition, the levels of abstraction and the respective *options* defined at each level such as 'wall following' and 'avoiding obstacles' are implemented differently.

2. The Problem

The problem dealt with in this paper is that of simple grid-world navigation. The map consists of square tiles arranged in a sequential manner in 2 dimensions, having mutually perpendicular walls, with the agent being able to occupy exactly one empty tile at a time. The map is structured in the form of intersecting corridors with randomly located obstacles, through which the agent must navigate, avoiding collisions. 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.

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.

Other issues involved are the dynamicity of the environment, such as varying wind blowing across the map, which may affect the agent's motion, and the stochasticity of world events and the agent's actions. Further the agent's sensors or effectors or both may be defective, causing noisy readings and erroneous output.

2.1 Problem Formulation

The entire state space and actions taken by the agent are formalized as a finite *Markov Decision Process* (MDP) [6]. 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) [11], 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* [13] which is nothing but a relation C : O → A, giving the optimal action to be chosen by the agent, corresponding to each observation.

2.2 Agent's Perceptors and Actuators

The agent has 8 sonars fixed horizontally in the each of the 8 principal directions. Each sonar sends out a beam which travels in that direction until it collides with a wall/obstacle and gets reflected. The agent makes use of the reflected beam to calculate the distance to the nearest obstacle in that direction. A sample set of sonar readings is shown in the following figure, with the blue tile indicating the position of the agent. Black tiles represent obstacles and numbers indicate the distance from the agent to the obstacle. Each such sonar reading constitutes an observation, in the simplest sense.

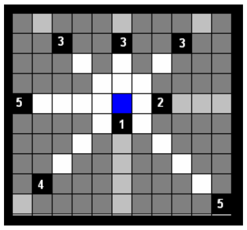

Figure 1. Agent's Perceptors

The actions available to the agent apart from sensing are Turn Left, Turn Right and Move Forward.

3. A Solution

The agent uses a short-term memory to remember recent observations and learns the optimal control policy using an on-policy Temporal Difference learning scheme, namely SARSA [13].

3.1 SARSA – A Temporal Difference Reinforcement Learning Algorithm

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 [13], this update takes place based on the previous Q-value.

An Є-greedy algorithm [13] 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 is an on-policy TD algorithm in the sense that the updates to Q-values are made at each step based on the action taken previously, and not the best possible action at that step.

The steps involved in the SARSA algorithm are as follows:

```
Initialize Q(s,a) arbitrarily
Repeat (for each episode)
    Initialize s
    Choose a from s using policy derived from Q (ie. ϵ - 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. ϵ - Greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha [ r + \gamma . Q(s',a') - Q(s,a) ]$
        $s \leftarrow s'$ ; $a \leftarrow a'$
    until s is terminal
```

SARSA 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.*

3.2 Memory and Hierarchy

As discussed already, the two major hurdles faced in this problem are that of scaling up to larger domains and perceptual aliasing. These can be overcome by organizing the learning process into different levels of a hierarchy, such that each level abstracts over minute details, which are present in the lower layers. Further by employing short-term memory instead of decisions based on single states, we can reduce the problem of perceptual aliasing.

Our state space now consists of histories, with each history representing a collection of recently observed states. 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 levels with each level containing a history of states previously observed at that level, instead of a flat, one-level state space.

A history can thus be defined as a sequence of the last *n* observations and the agent's memory is thus a collection of all such histories that have been encountered in the past. The agent's memory is stored in the form of a tree with the Q-values stored at each leaf node. The memory is padded with blank states until it reaches a minimum usable size.

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 other 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* [2] can be used. However, in this paper we restrict ourselves to memories of fixed length only.

Figure 2. shows a sample memory tree obtained during an agent's learning process. Successive states of a history are stored along a chain from root to leaf, with the leaf containing the Q-value for the history.

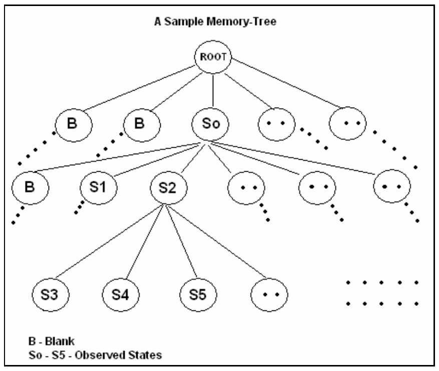

Figure 2. A sample Memory Tree

The above tree has memory size = 3 states and a blank state *B* is used to pad the memory with empty observations until it reaches the full size. Every path in the tree from Root to leaf represents a history of observations perceived by the agent. Thus the chain consisting of Root-So-S2-S5 represents the history wherein the last 3 observations were S5, S2 and So respectively, in that order.

Next, we build a hierarchy of levels for navigation, wherein each level of the hierarchy uses such a memory tree to store the observations perceived in that level.

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* [14] consisting of a temporally extended combination of these actions.

3.3 Implementation

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

A total of 13 different are intersections possible, 4 of which are shown below. Others can be obtained from rotation/reflection of these figures.

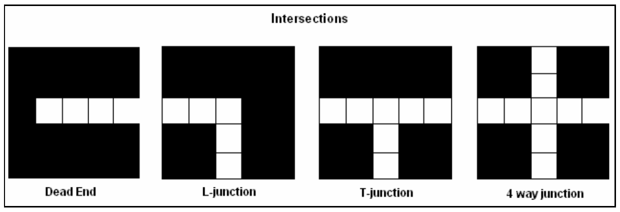

Figure 3. Types of intersections

Figure 3. shows simple corridor intersections with uniform thickness and having no obstacles, which is an elementary case. The actual map may contain corridors of varying thickness with randomly located obstacles.

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

Training:

The *option* used in the lower level must be capable of navigating within a corridor/hallway avoiding any obstacles that may be present there. For this purpose the agent is first trained in a Training Corridor, using the same Memory-Based SARSA algorithm, so that it learns to navigate within a corridor effectively. Once this is learnt, the same option is used throughout the map in level 2 of the hierarchy.

Figure 4. shows a sample training corridor where the green tile represents the start state and the red tile, the end state.

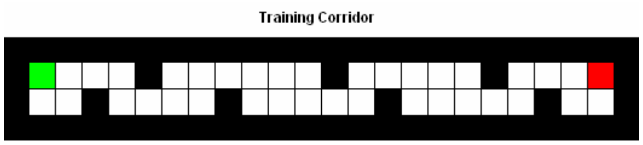

Figure 4. Training Corridor

The agent is repeatedly trained in such a training corridor and as the agent learns, the observation histories and corresponding Q-values are entered into the memory tree. This knowledge is then used by the agent while navigating the main map.

Learning and Updating Q-functions:

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

Level 1:

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

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

Level 2:

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

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.

4. Testing and Analysis

The algorithm explained in this paper for navigating partially observable domains has been tested comprehensively across a wide range of simulated maps, and the results obtained on two such maps are displayed in the following section.

These maps represent the actual physical structure of the world, and are not accessible to the agent. The agent's observation of a state on the map consists of only the 8 readings that it acquires as input from its sonars. Based on a history of such readings and the reward obtained, the agent learns a control policy by updating the corresponding value functions in its memory.

Also, it is observed that occasional errors in the sonar and noisy readings do not affect the learning process in the long run since they are not used directly by the agent for constructing a map of the world. Hence this design consisting of sonars is feasible for real-time implementation.

4.1 Results Obtained

Map 1:

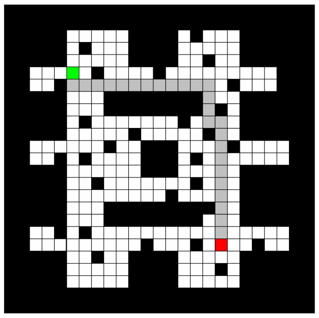

Figure 5. Map 1 – Without Obstacles

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

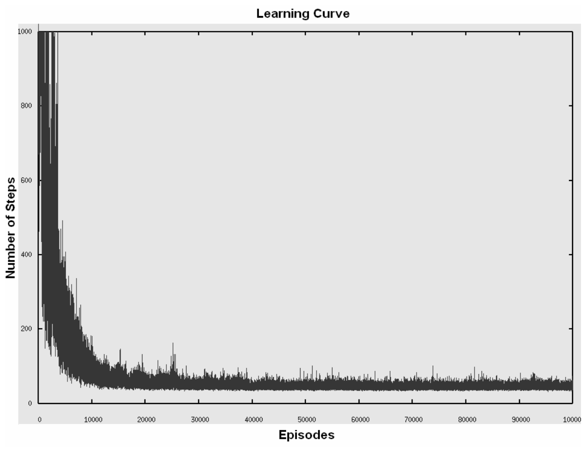

Figure 6. Map 1 – Learning Curve

Figure 6. shows that the Hierarchical Memory Based approach learns significantly well in the case of a corridor without obstacles (ie. Map 1).

Map 2:

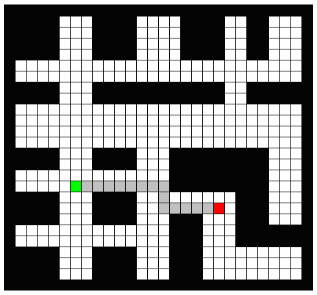

Figure 7. Map 2 – With Obstacles

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

In Map 2, we have compared the Hierarchical Memory Based learning algorithm with a flat, memory-less algorithm and obtained the following results:

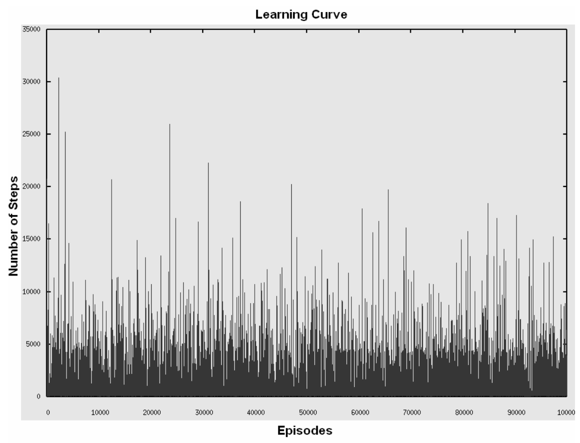

Figure 8. Map 2 – Learning Curve (Without Memory)

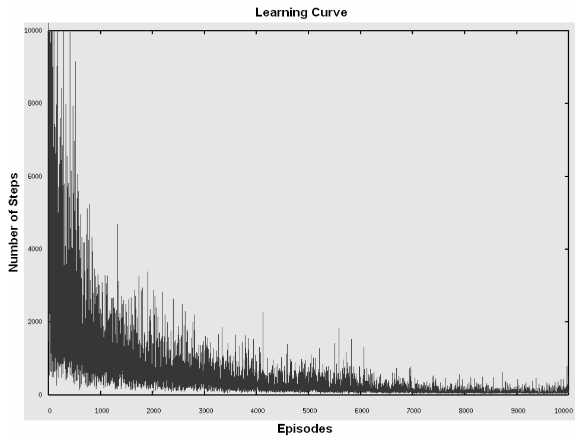

Figure 9. Map 2 – Learning Curve (With Hierarchical Memory)

In Figure 8. 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. However in Figure 9. we notice that the agent learns the optimal policy for navigation.

The agent is thus able to learn the optimal control policy efficiently in both maps 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.

5. Conclusion

5.1 Hurdles faced in the Implementation

i. It was not possible to uniquely determine a state as belonging to Level 1 or Level 2 of the hierarchy.
ii. Moving the agent out of intersections could not be entirely automated and some manual prodding was necessary.
iii. The learning process would be disrupted if the goal state was hidden inside corridors.
iv. The agent had problems in navigating around irregular-shaped obstacles until it was fully trained.

5.2 Optimizations

The following list of optimizations can be applied to the above implementation, in order to improve the accuracy and efficiency of the learning process.

i. Using an implementation of variable length memory such as U-Trees, instead of the fixed-size memory that is being used at present.
ii. Implementing goal regression and prioritized sweep.
iii. Addition of eligibility traces to speed up learning.
iv. Exploration vs Exploitation trade-off : Decaying the value of ϵ gradually to obtain optimal solutions after learning has stabilized .
v. Incorporating transformations in histories to exploit symmetry and other similarities between histories.
vi. Use of a closest match algorithm to determine a match from the memory tree with highest degree of closeness to the history.

5.3 Computational Issues

The technique of Reinforcement Learning used in this paper is strictly iterative and as in any such problem, the efficiency of learning is greatly dependent on the complexity of the map.

However, when compared to a one-level learning scheme, Hierarchical learning requires fewer states to be remembered because of its two-level organization. This greatly reduces the space complexity of the algorithm and in turn the search-time and time complexity, the extent of this reduction being dependent on the definition of levels of the hierarchy.

Thus we infer that the Hierarchical Memory-Based approach performs significantly better than the flat state-based approach for navigating in partially observable domains, in terms of the learning speed as well as the efficiency.

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, staff and students of the Department of Computer Science & Engg., IIT Madras, for their support and encouragement.

REFERENCES

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

[2] 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.

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

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

[5] Leslie Pack Kaelbling et al, "Hierarchical Solution of Markov Decision Processes using Macro-actions", *Proceedings of the Fourteenth International Conference on Uncertainty In Artificial Intelligence,* 1998.

[6] Richard Bellman, "Dynamic Programming", Princeton University Press, 1957.

[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] Brie Finger & Jessica Fisher, "Nomad 200", Robotics – Lab Write-up, Harvey Mudd College, 2003.

[9] Thomas. G. Dietterich, "The MAXQ Method for Hierarchical Reinforcement Learning", 1998 International Conference on Machine Learning.

[10] Ronald Edward Parr, "Hierarchical Control and Learning for Markov Decision Processes", PhD. thesis. December, 1998.

[11] Georgios Theocarous, K. Rohanimanesh, and Mahadevan, S. "Learning Hierarchical Partially Observable Markov Decision Processs for Robot Navigation". In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2001.

[12] Georgios Theocarous, K. Rohanimanesh, Mahadevan, S. "Learning Hierarchical Partially Observable Markov Decision Processes for Robot Navigation".

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

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

[15] Michael L. Littman, Richard S. Sutton and Satinder Singh, "Predictive Representations of State", in Advances in Neural Information Processing Systems 14, pages 1555—1561, 2002.

[16] Richard D. Smallwood and Edward J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon", Operations Research, 21:1071--1088, 1973.