Multi-agent learning in mobilized ad-hoc networks

Yu-Han Chang
CSAIL, MIT
Cambridge, MA 02139
ychang@csail.mit.edu

Tracey Ho
LIDS, MIT
Cambridge, MA 02139
trace@mit.edu

Leslie Pack Kaelbling
CSAIL, MIT
Cambridge, MA 02139
lpk@csail.mit.edu

Abstract

In large, distributed systems such as mobilized ad-hoc networks, centralized learning of routing or movement policies may be impractical. We need to employ multi-agent learning algorithms that can learn independently, without the need for extensive coordination. Using only a simple coordination signals such as a global reward value, we show that reinforcement learning methods can be used to control both packet routing decisions and node mobility.

Introduction

Mobile ad-hoc networking is emerging as an important research field with a number of increasingly relevant real-world applications, ranging from sensor networks to peer-to-peer wireless computing. Researchers in AI and machine learning have not yet made major contributions this growing field. It promises to be a rich and interesting domain for studying the application of learning techniques. It also has direct applications back to AI, for example in the design of communication channels for groups of robots. We introduce mobilized ad-hoc networks as a multi-agent learning domain and discuss some motivations for this study. In the work done thus far, we applied standard reinforcement learning techniques to two distinct problems within this domain: packet routing and node movement. Using relatively straightforward adaptations of these methods, we are able to demonstrate good empirical results.

Mobile ad-hoc networks have not traditionally been considered a multi-agent learning domain partly because most research in this area has assumed that we have no control over the node movements, limiting research to the design of routing algorithms. Each node is assumed to be attached to some user or object that is moving with its own purpose, and routing algorithms are thus designed to work well under a variety of different assumptions about node mobility patterns.

However, there are many applications in which we can imagine that some of the nodes would have control over their own movements. For example, mobile robots might be deployed in search-and-rescue or military reconnaissance operations requiring ad-hoc communication. In such cases it may be necessary for some nodes to adjust their physical positions in order to maintain network connectivity. In these situations, what we will term *mobilized* ad-hoc networks becomes an extremely relevant multi-agent learning domain. It is interesting both in the variety of learning issues involved and in its practical relevance to real-world systems and technology.

There are several advantages gained by allowing nodes to control their own movement. Stationary or randomly moving nodes may not form an optimally connected network or may not be connected at all. By allowing nodes to control their own movements, it is clear that by intelligently controlling these movements, we could achieve better performance for the ad-hoc network. One might view these controllable nodes as "support nodes" whose role is to maintain certain network connectivity properties. As the number of support nodes increases, the network performance also increases. Given better movement and routing algorithms, we can achieve significant performance gains.

It is important to note that there are two levels at which learning can be applied: (1) packet routing and (2) node movement. We will discuss these topics in separate sections, devoting a large portion of our attention to the more difficult problem of node movement. Packet routing concerns the forwarding decisions each node must make when it receives packets destined for some other node. Node movement concerns the actual movement decisions each node can make in order to optimize the connectivity of the ad-hoc network. Even though we will use reinforcement learning techniques to tackle both these problems, they must be approached with different mindsets. For the routing problem, we focus on the issue of online adaptivity. Learning is advantageous because it allows the nodes to quickly react to changes in network configuration and conditions. Adaptive distributed routing algorithms are particularly important in ad-hoc networks, since there is no centrally administered addressing and routing system. Moreover, network configuration and conditions are by definition expected to change frequently.

On the other hand, the node movement problem is best handled off-line. Learning a good movement policy requires a long training phase, which would be undesirable if done on-line. At execution time, we should simply be running our pre-learned policy. We treat the problem as a large partially-observable Markov decision process (POMDP) where the agent nodes only have access to local observations about the network state. This partial observability is inherent to both the routing and movement portions of the ad-hoc networking problem, since there is no central network administrator. Even with this limited knowledge, learning is useful because it would otherwise be difficult for a human designer to create an optimized movement policy for each network scenario.

Related Work

We draw inspiration for this work from two different fields: networking and reinforcement learning. In the networking literature, some work on the effect of node mobility in ad-hoc networks has been done for applications in which movement and topology changes are on the time-scale of packet delivery. Nodes are then able to act as mobile relays physically transporting packets from one location to another. Grossglauser and Tse [2001] analyze a strategy in which source nodes send packets to as many different nodes as possible, which store the packets and hand them off whenever they get close to the intended destination nodes. Li and Rus [2000] consider a scenario in which mobile hosts make deviations from predetermined trajectories to transmit messages in disconnected networks. Chatzigiannakis et al [2001] consider the case where a subset of mobile nodes are constrained to move to support the needs of the protocol, and act as a mobile pool for message delivery.

Our work is in a different setting, in which topology changes are on a much longer time scale than packet delivery delay constraints. Nodes move in order to form and maintain connected routes, rather than to physically deliver packets in a disconnected network. Routing is thus an important aspect of our algorithms that influences and is informed by movement decisions. Perkins and Royer [1997] and Johnson and Maltz [1996] examine routing in mobile ad-hoc networks where no control over node movements is assumed, while Chen et al. 2001 deal with the issue of preserving a connected topology with only some portion of nodes awake at any one time.

From the reinforcement learning community, there has been some interest in applying learning techniques to improve network performance. Such adaptive algorithms may be better able to perform well under widely varying conditions. Boyan and Littman [1994] applied reinforcement learning techniques to the problem of routing in static networks. They showed that a simple adaptive algorithm based on the Q-learning algorithm (Watkins 1989) can out-perform a shortest-paths algorithm under changing load and connectivity conditions. Peshkin [2002] used policy search rather than Q-learning on the same problem, which allowed the system to search a richer policy space. We apply Q-learning to the case of mobile networks, where node connectivity is constantly changing.

A major problem in the networking domain is the high degree of partial observability. From the perspective of any one node in the network, most of the rest of the network state cannot be discerned. The node can only rely on messages passed to it from its neighbors to glean information about the rest of the network, and to compound the issue, we would like to minimize such informational messages since they only contribute to network congestion. Thus, we will have to deal with partial observability in creative ways.

There are several other related challenges that we face. For example, as we train the nodes, the behavior of the overall network changes as the nodes learn to use better policies. A number of other authors have developed techniques to deal with learning situations where the environment changes over time. In our case, the network environment may change over time as the agents learn and move to new positions or assume new behaviors. In slowly varying environments, Szita et al. [2002] provide a specialization of Littman and Szepesvári's [1996] techniques for generalized MDPs, showing that Q-learning will converge as long as the variation per time step is small enough. In our case, we attempt to tackle problems where the variation is much larger. Choi et al. [1999] investigate models in which there are "hidden modes". When the environment switches between modes, all the rewards may be altered. This works if we have fairly detailed domain knowledge about the types of modes we expect to encounter. For variation produced by the actions of other agents in the world, or for truly unobservable environmental changes, this technique would not work as well. Auer et al. [1995] show that in arbitrarily varying environments, we can craft a regret-minimizing strategy for playing repeated games. Mannor and Shimkin [2001] extend these results to certain stochastic games. These results are largely theoretical in nature and can yield fairly loose performance bounds, especially in stochastic games. Rather than filtering the rewards as we will do, Ng et al. [1999] show that a potential function can be used to shape the rewards without affecting the learned policy while possibly speeding up convergence. This assumes that learning would converge in the first place, though possibly taking a very long time. Moreover, it requires domain knowledge to craft this shaping function.

Problem overview

Our mobilized ad-hoc network consists of one or more source nodes, one or more receiver nodes, and a number of other wireless nodes within a constrained area. All nodes are independently initialized according to a uniform distribution over this area. The sources generate packets at a constant rate and move according to a random way-point model. The aim of all nodes other than the sources and receivers is to transmit the maximum possible number of packets from the sources to the receivers. Some of these nodes can move so as to aid transmission, while the rest are stationary.

Performance is measured by the proportion of packets successfully transmitted to the receivers. When inter-node link capacities are limited and buffer sizes are finite, packet drops may occur due to buffer overflow, thereby decreasing network performance. When inter-node link capacities are sufficiently large compared to the source packet generation rates, an equivalent performance metric is the average proportion of sources connected to the receivers over time.

The routing problem: Q-Routing

Q-routing (Boyan & Littman 1994) is an adaptive packet routing protocol for static networks based on the Q-learning algorithm, which we adapt for use in mobile ad-hoc networks. It is essentially a version of the distributed Bellman-Ford algorithm. The algorithm allows a network to continuously adapt to congestion or link failure by choosing routes that require the least delivery time. When a route becomes congested or fails, Q-routing learns to avoid that route and uses an alternate path. Due to its adaptive nature, we might expect that Q-routing would also work well in the mobile ad-hoc setting.

Q-routing is a direct application of Watkins' Q-learning (Watkins 1989) to the packet routing problem. Each node in the network runs its own copy of the Q-routing algorithm. A node x faces the task of choosing the next hop for a packet destined for some receiver node d. Using Q-routing, it learns the expected delivery times to d for each possible next hop y, where each possible next hop y is a neighbor node connected to x by a network link. Formally, Q-routing keeps Q-tables Q^x for each node x and updates these tables at each time period t as follows:

$$Q^x_t(d,y) = (1-\alpha)Q^x_{t-1}(d,y) + \alpha(b^x_t + \min_z Q^y_{t-1}(d,z)),$$

where $0 < \alpha < 1$ is parameter that controls the learning rate, and b_t is the time the current packet spent on the buffer or queue at node x before being sent off at time period t.

Q-learning estimates the $value$ or $cost$, V, associated with each state d, with $V = \min_z Q^x(d,z)$. In our case, the value of a state is the estimated time for delivery of a packet from the current node x to destination d via node z. Once the nodes have learned the values associated with each state-action pair, they simply execute a greedy policy to behave optimally. When a node receives a packet for destination d, it sends the packet to the neighbor y with the lowest estimated delivery time $Q^x(d,y)$.

Adapting Q-routing to the mobile ad-hoc network routing domain is fairly straightforward. Neighbor nodes are defined as the nodes within transmission range. The main difference is that the neighbor nodes y may appear and disappear quite frequently due to node mobility. When a node y moves out of range, we set the estimated delivery time to d via y to ∞; i.e., $Q^x(d,y) = \infty$. When a node y moves into range, we optimistically set the estimated time to 0; i.e., $Q^x(d,y) = 0$. This optimistic bias encourages exploration. That is, node x will always try sending packets via a node y that has just come into range. If this action results in a high estimated delivery time, then node x will quickly revert to its original behavior since $Q^x(d,y)$ will quickly be updated to its true value. On the other hand, if this action results in a good delivery time, then node x will continue to send packets via node y.

Empirical results for routing

Our empirical results give an indication of the power of using adaptive learning techniques in designing movement and routing algorithms for mobilized ad-hoc networks. The setup is described in Section . There are source, receiver, and support nodes in a square grid, usually 30x30 in size. Each node has a transmission range of 6. The support nodes may either be fixed stationary nodes or mobilized agent nodes.

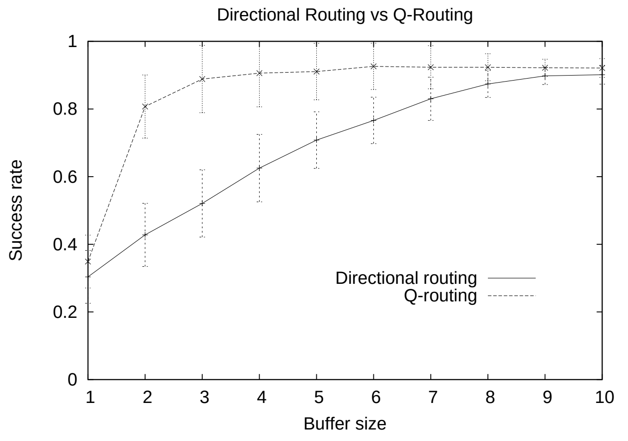

Figure 1: A comparison of directional routing vs Q-routing in a network with 10 sources, 15 mobile agents, and one receiver. Simulations were run over 20 initialization positions and 5 source movement scenarios for each different initialization. For each buffer size, averages over all of these trials are depicted, with error bars denoting one standard deviation.

There are two time scales: one for transmission and one for movement. During each movement time step, the node can choose one of its movement actions and perform 10 packet transmissions. Each packet transmission is the result of a Q-routing decision and update. Sources generate packets every two transmission time steps, and the number of packets received by a node in any time period is only limited by the node's buffer size.

To evaluate the performance of Q-routing, we implement a global-knowledge routing policy that is given information about the receiver location. This is done for comparison purposes only; in reality nodes generally do not have access to such information. With this information, nodes can route each packet towards the correct destination by forwarding the packet to a neighboring node that is closest to being in the direction of the receiver. We call this our *directional routing* policy. Specifically, our implementation forwards a packet to the neighbor that is closest to being in the direction of the receiver, up to a maximum deviation of 90 degrees. If no such neighbor exists, then the packet is dropped.

We compared Q-routing with directional routing under a variety of different scenarios. In almost all cases, Q-routing performs better than directional routing. Especially when the nodes are limited by small buffer sizes, Q-routing performs significantly better. Results for a typical set of network scenarios are shown in Figure 1. This is due to the fact that Q-routing will create alternate paths to the receiver as soon as a path becomes congested. Thus, packets will be less likely to be dropped due to buffer overflow caused by path congestion or limited buffer sizes since alternate paths will be constructed. In directional routing, on the other hand, often certain paths will become overloaded with traffic, causing significant packet drop.

Q-routing outperforms directional routing even in cases

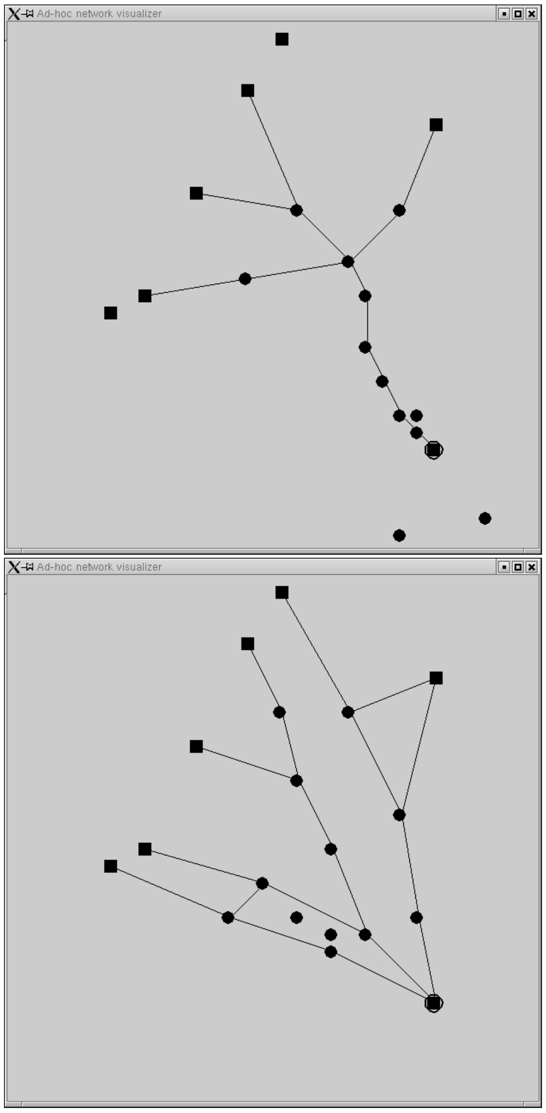

Figure 2: (Top) Using the directional routing policy, packets often become congested on the trunk of the network tree. (Bottom) Using Q-routing on the same experimental setup (note that the source and receiver nodes are in the same position as the both figures), the mobile nodes in the ad-hoc network spread out to distribute packet load. Sources are shown as squares, mobile nodes are circles, and the receiver is an encircled square. Both figures show the simulator after 10000 periods, using the same initialization and movement files.

where buffer size is not a direct constraint, such as the case shown in Figure 1 where the buffer size is 10. This illustrates the underlying problem of network capacity. Since the total source packet generation rate exceeds the transmission rate of any one inter-node link, directional routing may still run into trouble with bottleneck links regardless of buffer size. The network capacity achieved using Q-routing is larger since more alternate paths are created around such bottlenecks. Furthermore, directional routing is unable to find circuitous paths from sources to the receiver. Since it only routes packets to neighbors that are in the direction of the receiver, any path that requires a packet to be forwarded away from the receiver for one or more hops will never be found.

These comparisons are done using a fixed movement policy we will call the *centroidal movement* policy. Under this policy, a node that is holding a connection will attempt to move to the centroid of its connected neighbors, which increases the likelihood of preserving these connections over time. If it is not holding a connection, then it simply moves about randomly searching for a new connection. Thus, the next hops determined by the routing policy strongly influence the direction of movement, since the next hops determine the node's connections to its neighbors.

Q-routing and directional routing result in very different configurations for the network when coupled with centroidal movement. Directional routing tends to form a network backbone, which usually comprises the most direct route to the receiver for a large portion of the source nodes. Other sources send packets towards this backbone, resulting in a tree-like network configuration, as shown in Figure 2. Q-routing, on the other hand, always seeks the shortest path towards the receiver, even when buffer sizes are not a constraint. This results in a fan-shaped network configuration, also shown in Figure 2, where each source has its own shortest path to the receiver as long as there are a sufficient number of mobile nodes to create these paths. From this observation, we can begin to see that there is an interplay between the choice of routing protocol and the movement policy of the mobile nodes.

Learning to move

The problem of learning a good movement policy is much more difficult. We wish to again apply reinforcement learning techniques to this problem. First of all, the problem of partial observability is much more pronounced than in the packet routing problem. For example, when the network is in a disconnected state, information about the global network topology is impossible to collect but would be important for determining movements that would optimize network connectivity. Moreover, the local observation space could still be quite large, depending on the observed variables we choose to encode. Secondly, the choice of action space is unclear. At the most basic level, the agents could move in any direction at a given speed. We will limit the action choices by designing more complex actions that incorporate domain knowledge. This section briefly outlines our application of Q-learning to learn a reasonable movement policy despite the fact that Q-learning generally fails in POMDPs.

We proceed by treating the observation space as our "state space". This can potentially lead to problems of aliasing, but in this first attempt, we simply try to choose our observed variables carefully in order to avoid this pitfall. Since the nodes communicate using a shared wireless medium, a node can "sniff" packets sent by neighbors to destinations other than itself. Thus, a node can detect the presence of neighbor nodes and their connections, even if it is not involved in any of these connections itself. Moreover, since the receiver nodes send back acknowledgement packets along these connections, our agent node can also collect statistics about these source-destination paths by sniffing these acknowledgement packets. Each agent node's observation space thus includes the number of neighbors, the number of connections it is currently holding, the number of nearby connections, and the maximum and minimum average hop lengths of these source-destination paths.

In order to constrain the learning process, we incorporate some domain knowledge into our design of an appropriate action space. For example, there is little need to train the nodes to avoid obstacles along their desired path of movement. We can pre-program the nodes with the necessary algorithm to do this. Learning is focused on higher-level action selection that is difficult to pre-program effectively. A subset of our action space is given in the table below.

Action	Description
stay	Stay put; don't change position.
direction	Head in a particular direction.
plug	Searches for the sparsest path and attempts to fill in the largest gap along that path.
leave	Leaves a path and becomes available for other actions.
center	Moves to the centroid of the neighbors to which it is connected.

Finally, the reward given to the nodes during each time period corresponds to the percentage of successful transmissions during that time period, which is available since we are conducting this training off-line. During online execution of the ad-hoc network, this network performance measurement would not be available since there would be no trainer that has access to the global state of the network. With these assumptions, we can construct a standard Q-learning algorithm that tries to learn good movement policies as the agents interact in simulation.

Movement policy comparisons

We evaluate the performance of our learning algorithm against the centroidal movement policy given in Section , a hand-coded policy that uses the same observation space as the learning algorithm, and a global-knowledge central controller. Under simulation, we give the central controller access to all the node, source, and receiver positions, which would usually be unavailable to the agent nodes. Since our learning algorithm only has access to local knowledge, the central controller's performance should approximate an upper bound for the learning algorithm's performance. Moreover, this performance bound may fluctuate over time as network conditions change depending on source movement scenarios.

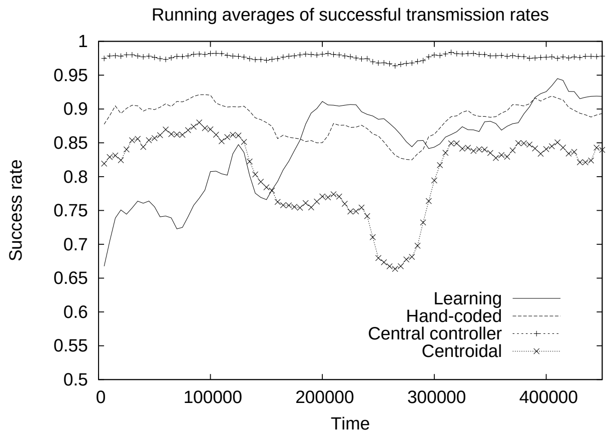

Figure 3: Graph showing the average performance of various movement policies over time in a typical scenario. The learning policy is shown during its training phase.

The central controller is designed to approximate an optimal movement policy given global knowledge about network conditions. It begins by identifying connected components among the stationary nodes. If a packet is received by any node of a connected component, then all nodes of that component can also receive the packet. However, if a packet's destination is in a different component, then the controller finds the shortest path to the destination component and recruits mobile nodes to help construct the necessary links needed to deliver the packet.

Figure 3 gives the average performance of a sample network scenario over time using each of these movement policies. As we can see, the learning algorithm does eventually learn a policy that behaves fairly well, but it never achieves the performance of the global knowledge controller. This is expected since the learner does not have access to the global network topology. On average, the learned policies perform slightly better than the hand-coded policy over large sets of network scenarios. However, in certain scenarios, it never learns to perform as well, possibly due to aliasing problems.

A simplified view

There are many further complications to learning in this mobilized ad-hoc networking setting. As mentioned earlier, the learning agent does not have a full view of the world – it cannot see the world state of agents that are far away or otherwise obscured. Furthermore, it certainly does not have a complete representation of the internal states of the other agents. This partial observability creates problems when the agent begins to learn about the world, since it cannot see how the other agents are manipulating the environment and thus it cannot ascertain the true world state. Aliasing may occur, a situation in which the agent cannot distinguish between one good state and one bad state because the observations it receives in both states are the same. This is an acute problem in partially-observed multi-agent learning, since the unobserved agents could have a tremendous effect on the value of a state.

A separate problem arises when we train multiple agents using a global reward signal. Even with full observability, the agents would need to overcome a credit assignment problem, since it may be difficult to ascertain which agents were responsible for creating good reward signals. Ideally the agent can recover the some true personal reward signal and learn using that signal rather than the global reward signal. One crude approach is for an individual agent to model the observed global reward signal as the sum of its own contribution (which is the personal reward signal on which it should base its learning) and a random Markov process (which is the amount of the observed reward due to other agents or external factors). With such a simple model, we can estimate both of these quantities efficiently using an online Kalman filtering process. Many external sources of reward (which could be regarded as noise) can be modeled as or approximated by a random Markov process, so this technique promises broad applicability. This approach is more robust than trying to learn directly from the global reward, allowing agents to learn and converge faster to an optimal or near-optimal policy, sometimes even in domains where convergence was once elusive.

The innovative aspect of our approach is to consider the reward signal as merely a signal that is correlated with our true learning signal. We propose a model that captures the relationship between the true reward and the noisy rewards in a wide range of problems. Thus, without assuming much additional domain knowledge, we can use filtering methods to recover the underlying true reward signal from the noisy observed global rewards.

Mathematical model

The agent assumes that the world possesses one or more unobservable state variables that affect the global reward signal. These unobservable states may include the presence of other agents or changes in the environment. Each agent models the effect of these unobservable state variables on the global reward as an additive noise process b_t that evolves according to $b_{t+1} = b_t + z_t$, where z_t is a zero-mean Gaussian random variable with variance σ_w. The global reward that it observes if it is in state i at time t is $g_t = r(i) + b_t$, where r is a vector containing the ideal training rewards $r(i)$ received by the agent at state i. The standard model that describes such a linear system is:

$$g_t = Cx_t + v_t, \quad v_t \sim N(0, \Sigma_2)$$
$$x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, \Sigma_1)$$

In our case, we desire estimates of $x_t = [r_t^T \quad b_t]^T$. We impart our domain knowledge into the model by specifying the estimated variance and covariance of the components of x_t. In our case, we set $\Sigma_2 = 0$ since we assume no observation noise when we experience rewards; $\Sigma_1(j,j) = 0, j \neq |S| + 1$, since the rewards are fixed and do not evolve over time; $\Sigma_1(|S|+1, |S|+1) = \sigma_w$ since the noise term evolves with variance σ_w. The system matrix is $A = I$, and the observation matrix is $C = [0 \quad 0 \ldots 1_i \ldots 0 \quad 0 \quad 1]$ where the 1_i occurs in the i^{th} position when our observed state $s = i$.

Kalman filters (Kalman 1960) are Bayes optimal, minimum mean-squared-error estimators for linear systems with Gaussian noise. The agent applies the following causal Kalman filtering equations at each time step to obtain maximum likelihood estimates for b and the individual rewards $r(i)$ for each state i given all previous observations. First, the estimate $\hat{x}$ and its covariance matrix P are updated in time based on the linear system model:

$$\hat{x}'_t = A\hat{x}_{t-1} \tag{1}$$
$$P'_t = AP_{t-1}A^T + \Sigma_1 \tag{2}$$

Then these a priori estimates are updated using the current time period's observation g_t:

$$K_t = P'_t C^T (CP'_t C^T + \Sigma_2)^{-1} \tag{3}$$
$$\hat{x}_t = \hat{x}'_t + K_t(g_t - C\hat{x}'_t) \tag{4}$$
$$P_t = (I - K_t C)P'_t \tag{5}$$

As shown, the Kalman filter also gives us the estimation error covariance P_t, from which we know the variance of the estimates for r and b. We could also create more complicated models if our domain knowledge shows that a different model would be more suitable. For example, if we wanted to capture the effect of an upward bias in the evolution of the noise process (perhaps to model the fact that all the agents are learning and achieving higher rewards), we could add another variable u, initialized such that $u_0 > 0$, modifying x to be $x = [r^T \quad b \quad u]^T$, and changing our noise term update equation to $b_{t+1} = b_t + u_t + w_t$. In other cases, we might wish to use non-linear models that would require more sophisticated techniques such as extended Kalman filters.

For the learning mechanism, we use a simple tabular Q-learning algorithm, since we wish to focus our attention on the reward signal problem. The agent follows this simple algorithm:

1. From initial state s_0, take some action a, transition to state i, and receive reward signal g_0. Initialize $\hat{x}_0(i_0) = g_0$ and $\hat{x}_0(|S|+1) = b_0 = 0$, since $b_0 = 0$.
2. Perform a Kalman update using equations 1-5 to compute the current vector of estimates $\hat{x}$, which includes a component that is the reward estimate $\hat{r}(s_0)$, which will simply equal g this time.
3. From the current state i at time t, take another action with some mix of exploration and exploitation; transition to state j, receiving reward signal g_t. If this is the first visit to state i, initialize $\hat{x}_t(i) = g_t - \hat{b}_{t-1}$.
4. Perform a Kalman update using equations 1-5 to compute the current vector of estimates $\hat{x}$, which includes a component that is the reward estimate $\hat{r}(i)$.
5. Update the Q-table using $\hat{r}(i)$ in place of r in a standard Q-learning update step; return to Step 3.

The advantage of the Kalman filter is that it requires a constant amount of memory – at no time does it need a full history of states and observations. Thus, we can run this algorithm online as we learn, and its speed does not deteriorate over time.

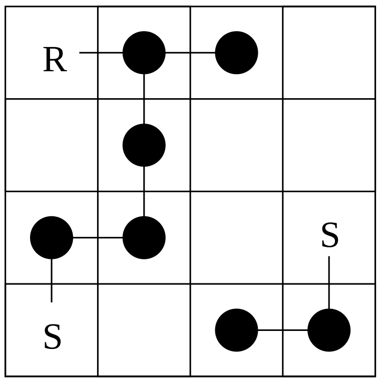

Figure 4: A snapshot of the 4x4 adhoc-networking domain. S denotes the sources, R is the receiver, and the dots are the learning agents, which act as relay nodes. Lines denote current connections. Note that nodes may overlap.

Empirical results

If the world dynamics match the linear model we provide the Kalman filter, then certainly this method will work well. Clearly this is not the case in our ad-hoc networking domain. However, in our previous work (Chang, Ho, & Kaelbling 2004), we considered several scenarios in which the model fit the environment more closely. Here, in order to study the adhoc networking domain using this technique, we simplify the domain by adapting it to the grid world. As discussed earlier, mobilized ad-hoc networking provides an interesting real-world environment that illustrates the importance of reward filtering due to its high degree of partial observability and a reward signal that depends on the global state. The nodes are limited to having only local knowledge of their immediate neighboring grid locations, and thus do not know their absolute location on the grid. They are trained using a global reward signal that is a measure of total network performance, and their actions are limited functions that map their local state to N, S, E, W movements. We also limit their transmission range to a distance of one grid block. For simplicity, the single receiver is stationary and always occupies the grid location (1,1). Source nodes move around randomly, and in our example here, there are two sources and eight mobile agent nodes in a 4x4 grid. This setup is shown in Figure 4, and Figure 5 shows a comparison of an ordinary Q-learner and the filtering learner, plotting the increase in global rewards over time as the agents learn to perform their task as intermediate network nodes. The graph plots average performance over 10 runs, showing the benefit of the filtering process.

Ongoing Work

We are continuing to explore the mobilized ad-hoc networking domain as a vehicle for further study of learning in cooperative multi-agent systems.

For larger versions of the networking domain, we may need different variants of our modeling and filtering framework. Currently we use a very simple linear model that contains only one degree of freedom and happens to conform to the Kalman filtering framework. However, we might imagine that there are many scenarios where actions taken by the other agents in the environment might affect the global reward in some other manner, such as the scenario described earlier where global performance increases over time due to learning.

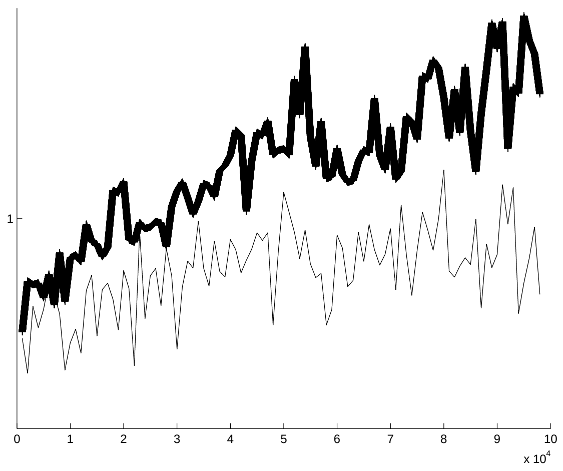

Figure 5: Graph shows average rewards (y-axis) in 1000-period windows as filtering (bold line) and ordinary (thin line) agents try to learn good policies for acting as network nodes. The filtering agent is able to learn a better policy, resulting in higher network performance (global reward). Graph shows the average for each type of agent over 10 trial runs of 100000 time periods (x-axis) each.

Secondly, we have assumed that the reward signal is deterministic. In domains where the stochasticity of the rewards approximates Gaussian noise, we can use the Kalman framework directly. In equation 1, v was set to exhibit zero mean and zero variance. Allowing some variance would give the model an observation noise term that could reflect the stochasticity of the reward signal. There are some cases which cannot be finessed so easily, though.

There are two potential remedies in this situation, which we are pursuing. One solution modifies the system equations so that the vector to be estimated represents the average reward over a time window, rather than a single deterministic value. Another alternative makes two passes over a history window. In the first pass, we do exactly the same as before, except that we also note the log-likelihood of each of our observations, based on the Kalman filter statistics. During the second pass, for each state that consistently exhibits unlikely observations, we split the state into one or more states, each corresponding to a different reward level. We then examine the average log-likelihood under this new model, and if it represents a significant improvement over the old model, we keep the split states.

We are also continuing to develop methods for dealing with macro-actions and semi-MDPs. Memory could be exploited to more effectively deal with the credit assignment problem. We might also explore the possibilities that can be achieved via specialized, limited inter-agent communication that is separate from the network packets being transmitted. While this would introduce network overhead, performance gains could be significant. We are also considering applying this method to a competitive ad-hoc networking domain variant, where adversial nodes may attempt to disrupt communications.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 1995. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*.

Boyan, J., and Littman, M. L. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in NIPS*.

Chang, Y.; Ho, T.; and Kaelbling, L. P. 2004. All learning is local: Multiagent learning in global reward games. In *Advances in Neural Information Processing Systems (NIPS03)*.

Chatzigiannakis, I.; Nikoletseas, S.; Paspallis, N.; Spirakis, P.; and Zaroliagis, C. 2001. An experimental study of basic communication protocols in ad-hoc mobile networks. In *5th Workshop on Algorithmic Engineering*.

Chen, B.; Jamieson, K.; Balakrishnan, H.; and Morris, R. 2001. Span: An energy-efficient coordination algorithm for topology maintenance. In *SIGMOBILE*.

Choi, S.; Yeung, D.; and Zhang, N. 1999. Hidden-mode Markov decision processes. In *IJCAI Workshop on Neural, Symbolic, and Reinforcement Methods for Sequence Learning*.

Grossglauser, M., and Tse, D. 2001. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM*.

Johnson, D., and Maltz, D. 1996. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers, Journal of Basic Engineering*.

Li, Q., and Rus, D. 2000. Sending messages to mobile users in disconnected ad-hoc networks. In *MOBICOM*.

Littman, M. L., and Szepesvári, C. 1996. A generalized reinforcement-learning model: Convergence and applications. In *Proc. of the 13th ICML*.

Mannor, S., and Shimkin, N. 2001. Adaptive strategies and regret minimization in arbitrarily varying Markov environments. In *Proc. of 14th COLT*.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th ICML*.

Perkins, C., and Royer, E. 1997. Ad-hoc on-demand distance vector routing. In *MILCOM Panel*.

Peshkin, L. 2002. Reinforcement learning by policy search. *Ph.D. Thesis, MIT*.

Szita, I.; Takacs, B.; and Lorincz, A. 2002. e-mdps: Learning in varying environments. *Journal of Machine Learning Research*.

Watkins, C. J. 1989. Learning with delayed rewards. *Ph.D. Thesis, University of Cambridge*.