Chapter 5

Machine Learning and WSNs

This Chapter highlights the importance of specific machine learning tools namely Support Vector Machines (SVM) and Reinforcement Learning (RL) to WSNs. The primary interest of learning techniques in the field of sensor networks is to reduce the amount of energy consumed by sensor nodes by reducing the amount of communication in the network. Learning techniques in sensor networks are attractive for the four following reasons. First, there usually exists a degree of redundancy in sensor network data. The redundancy stems from the fact that geographically close sensors are likely to collect similar measurements, and measurements taken at two consecutive instants are also likely to be similar. Learning allows detecting these redundancies and represents in a more compact way. Accordingly in the first part of this chapter we study application of SVM to the localization problem and analyze that SVM indeed results in energy savings. In the second part we study application of Reinforcement Learning to routing in WSNs.

5.1 Support Vector Machines in WSNs

In a wireless sensor network, the data collected from different sensor nodes is then linked together for a meaningful interpretation & application. This often requires relating sensor data to its physical location. Localization forms the very first step in the sensor network based applications.
The majority of the localization algorithms use information like time of arrival of signals (using directional antennas) or Received Signal Strength Indicator (RSSI) measurements for localization.

Typically, Received Signal Strength Indicator is not the most accurate indicator for device location, and is subject to various multi-path and fading effects, which may be varying irregularly over time. Other parameters like the angle of arrival and timing based indicators provide more accurate observations relating to the location of the node. However, they require a dedicated hardware co-located with the sensor node for logging and providing this information. In case of RSSI, the information is already available to the sensor node and no extra hardware is required. This is particularly useful for energy constraint nodes.

The traditional localization approaches use range estimate as the first step in the localization process. Using signal spreading models, an approximation is formed about the distance between a pair of points. However, accuracy of such scheme is limited by the ranging errors. Range free localization from RSSI data is a non-standard problem and there have been only few attempts so far in this direction. In [253], authors described a simple nearest neighbor classification algorithm to obtain coarse localization of objects. A kernel based localization algorithm is proposed in [254]. Motivated by the nature of the problem and some recent work in this area, we see the sight of the use of a learning algorithm for solving this problem. We present an approach using SVM to the problem.

5.1.1 Localization in AdHoc Sensor Networks
**The Scenario**

Let $X_1, X_2, \ldots, X_m, X_{m+1}, \ldots, X_{m+n}$ be the locations of $m+n$ sensor nodes with $\mathbb{R}^3$. We assume that the position of the first $m$ nodes is unknown and represents this state by $X_u = \{X_1, X_2, \ldots, X_m\}$. Also, we assume that the position of the 'n' nodes are known and represent this state by $X_k = \{X_{m+1}, X_{m+2}, \ldots, X_{m+n}\}$. We call the nodes in $X_k$ as beacon nodes.

In general $n \ll m$. For every pair of nodes $(i, j)$ we are given the signal strength that sensor $j$ receives from node $i$. The problem is to recover the positions of all nodes $\in X_u$.

**Brief description of Algorithm**

Given any pair of nodes $(X_i, X_j)$, we train SVM to assign label 1 to it if $\|X_i - X_j\|^2 < R_o$ or $-1$ elsewhere, where the choice of $R_o$ is up to us. Training is done on the set of beacon nodes. We form a feature vector which is robust to signal variations. Once we have this pairwise classification output, we map the SVM output to probability by fitting the data to a logistic regression as described in [255]. Our intuition is that a pair of points, which are very close, will be assigned label 1 with probability close to 1 and points, which are very far apart, will be assigned label 1 with probability very close to 0. We then use this pairwise probability to estimate the position of unknown nodes by various schemes as described in the next section.

**Detailed Description of the Algorithm**

**A. Training SVM and Classification**
Consider a pair of nodes \((i, j)\). Where the intuition is that two nodes are close in space if they receive the similar signal from all other nodes. Thus, the feature vector for nodes \((i, j)\) is given by \(\emptyset (i, j)\) where it’s \(k^{th}\) element \(\emptyset (i, j)_k = \exp \left( \frac{||S_{ik} - S_{jk}||^2}{\tau} \right)\).

This choice of feature vector makes the system robust to fading that is commonly observed in wireless signals. So, if there is some obstruction in the line of sight between node \(i\) and node \(k\) and if node \(j\) is close to node \(i\) then \(S_{jk}\) will also show the variations similar to that observed in \(S_{ik}\). Note that to reduce the training complexity, we can consider only those nodes \(k\) for which at least one of \(S_{ik}\) or \(S_{jk}\) have value greater than some pre-determined threshold, though, we haven’t done this in our current implementation. With this choice of features, we train the SVM to answer if the distance between a pair of nodes is less than \(R_o\). All possible pairs of beacon nodes are used for training. We implement this using SMO algorithm with L1 regularization [256]. Once trained, we store the classification output in matrix \(Y\) and value of SVM before threshold in a matrix \(S\) where \(y_{ij} = 1\) if SVM output for nodes \((i, j)\) is \(> 0\) and \(y_{ij} = -1\) otherwise. Similarly, \(s_{ij} = w' \emptyset (i, j) - b\) denotes SVM output for nodes \((i, j)\) before threshold.

### B. Mapping SVM output to probability

We use the approach given in [254] to map SVM to probability. This is done by first fitting the parameters of a logistic regression function \(g(s) = 1/(1+exp (A s + B))\) and then evaluating \(g(s_{ij})\) for each \(s_{ij} \in S\). This gives us a matrix \(P\) where \(P_{ij} = g(s_{ij}) = \exp \left( \frac{||S_{ik} - S_{jk}||^2}{\tau} \right)\)
\[ P[\|X_i - X_j \|^2 < R_0] \]. For fitting the parameters \(A\) and \(B\), we form following log likelihood function and maximize it with respect to \(A\) and \(B\).

\[(A, B) = \arg \max_{A, B} \sum f(t(f) \log(g(f)) + (1 - t(f)) \log(g(f))),\]

where \(f\) is SVM output for a pair of beacon nodes after it has been trained on the beacon nodes and \(t(f)\) is the corresponding label which takes value 0 and 1 instead of -1 and 1. In order to get an unbiased estimate of the probability, we split the training data into three parts, trained SVM on all combinations of two out of three, and used the output of that SVM on the third part and corresponding labels of third part for fitting logistic regression parameters.

C. Estimating locations

We use the matrix \(P\) obtained earlier for estimating the locations. One possible approach is to write the position of node \(i\) as the weighted sum of its neighbors.

\[ X_i = \sum_{j \in N(i)} w_{ij} X_{ij} \]

\[ (2) \]

where \(N(i)\) denotes the neighbors of node \(i\) which can be taken as those nodes \(j\) for which entries \(C_{ij}\) is 1. An obvious method to choose \(w_{ij}\) is to take them in proportion to \(p_{ij}\). A sequential algorithm can be formed as follows: We first find the position of nodes, which has maximum number of neighbors with known locations (for the very first pass, the node with maximum number of beacon nodes in its neighborhood). We compute the location of this node as the weighted sum of locations of its neighbors with known locations. As further node locations becomes known, they help in locating other nodes. This scheme is very simple but is prone to error propagation.

Fig. 5.1 shows the outcome of such scheme. An alternative would be to use joint detection that tries to minimize
Figure 5.1: The process of locating other nodes as new ones become known

If \( P_u, Qu, Re \) and \( P_k, Q_k, R_k \) are column vectors denoting the \( x, y \) and \( z \) co-ordinates of nodes in \( X_u \) and \( X_k \) respectively, and if we partition weight matrix \( W \) as \( W = [W_u W_k] \) then it can be shown that minimizing (3) gives

\[
P_u = (W_u^T W_u - W_u^T W_k + I)^{-1} (W_k P_k - W_k^T X_k) \tag{4}
\]

and similar expressions can be obtained for \( Q_u \) and \( R_u \). We call this as MinNorm\(_2\) scheme. Though, we are doing a joint detection, when positions of all the nodes are unknown, a trivial solution for (3) is given when all the nodes are at single point. In presence of some beacon nodes, nodes won’t stick to one point but they will still have the tendency of converging towards the center since number of beacon nodes are very small compared to unknown nodes as shown in fig. 5.2. An alternative would be to
minimize first norm instead of second norm, but this improves the results only by a small margin.

Figure 5.2: Minimizing the second norm- $\text{MinNorm}_2$

A better solution is given by an application of Laplace Eigen maps [257]. This involves minimizing the cost function

$$S_{LE} = \sum_{i,j} w_{i,j} ||Z_i - Z_j||^2$$  \hspace{1cm} (5)

subject to constraints

$$\sum_i Z_i = 0; \ and \ \sum_i ||Z_i||^2 = 1;$$  \hspace{1cm} (6)

The constraints remove the translation ambiguity and the tendency to put all the points at origin. We take the weights $w_{i,j} = P_{i,j}$. The intuition behind using this optimization problem is that nodes for which $P_{i,j}$ is close to 1 are likely to be close and nodes for which $P_{i,j}$ is close to zero are likely to be far apart. Let W denote the
weight matrix. We define $u_i = \sum w_{ij}$ and $L = \text{diag} \ [u_1, \ldots, u_n] - W$. Let $(\lambda_k, bfv_k)$ be the Eigen value decomposition of $L$ which are arranged in the increasing order by magnitude of eigenvalues. Then the optimal lowest cost 3-dimensional solution to (5) is given by $Z_i = [v_2 (i), \ldots, v_4 (i)]^T$.

However, the co-ordinate system for $Z_i$ and $X_i$’s are different. We map $Z_i$ to $X_i$ by the following way. We assume that in a noise free environment $X_i$ are obtained from $Z_i$ by an affine transformation

$$X_i = AZ_i + b \quad (7)$$

where $A$ is $3 \times 3$ matrix and $b$ is a $3 \times 1$ matrix. We estimate $A$ and $b$ by posing a least square error problem over beacon nodes estimate given by $X_i$ s and $Z_i$ s.: 

$$(A, b) = \arg \min_{A, b} \sum_{i \in X_k} \|AZ_i + b - X_i\|^2 \quad (8)$$

Once we know $A$ and $b$, we map back to the original coordinate system by using (7). The results are shown in fig. 5.3.
A similar approach based on nodes connectivity has been tried in [258] but our approach differ in a sense that we have a definite scheme for choosing weights by using SVM probability values and we propose the use of an Affine transformation to go back to the original co-ordinate system.

### 5.1.2 Simulation Results

We test the validity of the algorithm by simulation. We use the signal model described in [254] for evaluating the performance. Each sensor location $x$ is assumed to receive from a sensor located at $x'$ a signal value following a fading channel model:

$$s(x, x') = exp\left(-\frac{\|x-x'\|}{\sigma_1}\right) + N(0, \sigma_2).$$

where $N(0, \sigma_2)$ is the independently generated Gaussian random variable with zero mean and variance $\sigma_2$. This has nothing to do with the choice of the feature vector and
A polynomial decay function was found to give similar results. Simulations are carried out for three different values of noise power $\sigma = 0.01, 0.05, 0.1$ while $\theta = 1.7$ is kept fixed. The percentage classification errors for these values of noise were found to be 2.82, 3.1, and 3.85 respectively. The parameters for fine localization error is presented in table 5-1 where the values are normalized, so they represent the error that would have occurred in an area of $1 \times 1$. As expected, the performance deteriorates with the increase in the noise power. Moreover, Laplace Eigen maps perform much better than the other schemes. Fig. 5.4 shows the outcome of a typical initial classification stage. Here the all the points, which are within $R_0$ distance from center node, are marked within the large circle. A blue point means a classification label 1 while a red point means a classification label $-1$. Figs. 5.1-3 show the outcome of the sequential, $\text{MinNorm}_2$ and Laplace Eigen maps schemes respectively for the worst case noise power of 0.1. Blue dots denote the true position of sensor nodes while red dots denote their estimated position.

![Figure 5.4: Output of a typical initial classification stage.](image-url)
The algorithm needs to be tested on actual data. It has been shown by researchers that the percentage classification error on data with few nodes is high as 30. This is mainly due to a very small number of nodes compounded by even a smaller fraction of beacon nodes. To evaluate the algorithm, a large network of actual sensor nodes would be required.

Conclusions on SVM application to Localization in WSNs

We present a new learning algorithm for Ad-hoc sensor network localization & show that mapping output of SVM to probability followed by application of Laplace Eigen maps and Affine transformation gives a good estimate of node locations from RxSigSI data analysis. The algorithm is particularly useful in dense sensor networks. Our contribution in the context of this research is to show a completely different approach to solving a difficult problem of localization from Received Signal Strength data. The implementation of this algorithm shows that a learning algorithm can be applied for this class of problems—localization in WSNs. A more immediate goal is to test the algorithm on a large actual sensor network. We would also like to apply thin plate spline transformation instead of Affine transformation to map back to the original coordinate system once we have the outcome of the Laplace Eigen maps.

Current implementation of SVM has high computational complexity for the training. Attempts can be made to reduce the complexity of the training phase by a good choice of Kernel, which doesn’t require computing feature explicitly, or by limiting the size of the feature vector. An analysis can be made to find better schemes for choosing the weights or to find the optimal weights. A distributed version of the algorithm is also being developed. Last but not the least, I would like to acknowledge and thank Professor Andrew Ng and his online Machine Learning course for the basic motivation on this aspect that is application of ML to WSNs.
5.2 Reinforcement Learning Approach for WSNs

The data processing capabilities and techniques are increasing exponentially. So modern engineering community has become increasingly reliant on sensor data to provide an accurate assessment of system behavior and performance. It is usually impractical to install sensing transducers in sufficient numbers because of the high cost of installing and maintaining data cables in large engineering systems. We thus find that wireless sensing systems, which can be deployed at less than one-tenth of the cost of traditional tethered systems, are being explored as a new interface between sensing transducer and data repository. Besides their low costs, wireless sensing networks (WSNs) have also shown great promise because of their ability to process sensor data locally at each wireless node.

The rft lrng iss aa sb-set aaf mchn lrng cncrnd wth lrng frm interactions bye trls & errs and hw tto bhve inn ordr tto achve aa gl. Imprtnt notns inn a rft lrng formln aas aa Mrkv Dcsn Prcss & iits rsuln cn bee summrzd aas flw:

Mrkv Prperty. Ann envrnmnt satisfs th Mrkv prperty iff th st sig cmpctly smmrztes th pst w/o degrdg th ably to prdct th futre. Iif th Mrkv prptry hlds, thn th RftL envrnmnt iis clld aa Mrkv Dcsn Prcss (MDP).

Mrkv Dcsn Prcss (MDP). Frmlly aa fnte MDP iss aa tple <S, A, T, R> whr S iss aa fnte st of envrnmnt states; A iss aa st aaf actns avlbl aat th agnt; T: S× A > π(S) iis th st transtn fn gvn fr ea ste & actn aa prob distr ovr stes; R: SxA is th rft fn tht indcts th rl-val
Partially Observable MDP (POMDP). The POMDP is a variant of the MDP in which the state of the environment is only partially visible to the learning agent. What are available are indirect, potentially stochastic observations of the environment state.

Val Fns:
Almost all learning algorithms are based on estimating either state-value or action-value functions. State-value functions $V$ reward the agent when starting in state $s$ and following policy $\pi$ thereafter. Action-value function $Q^\pi(s, a)$ estimates the expected reward when it performs a given action in a given state & following policy $\pi$ thereafter.

5.2.1 Formulation as Reinforcement Learning Problem
This problem can be designed as a reinforcement learning problem in many different ways. Probably, the most straightforward approach is to think of an agent which will control the entire network and assign nodes to jobs as new jobs appear and are completed.

Advantages of formulation view include:
1) It can be modeled as a Markov decision process (MDP).
2) It is quite easy to analyze learned policies.

Disadvantages of formulation view include:
1) Delayed reward.
2) Large state space.
3) Difficulty in handling new tasks.

We can formulate a WSN with an agent controlling node assignments as a MDP (let us call that $M$) by making some simplifying assumptions. $M$ can be described by the 4-tuple as $M = (S, A, P, R)$ where $S$ represents a finite state space, $A$ represents a finite set of actions, $P$ are transition probabilities, and
\( R \) is the reward function. Firstly, we model the WSN as a set of homogeneous nodes. This is not the case with real networks, because some nodes have higher costs of communication due to their physical location.

- State in this formulation corresponds to the status of each computing job as seen by the WSN. Although the number of jobs \( J \) may keep on changing over time, we can upper bound it to create a set of finite state space. For each job number \( i \in \{1, 2, 3, 4, \ldots, J\} \) we will associate five values namely: type of task \( k_i \), total communication cost accrued thus far \( c_i \), number of nodes assigned to this task \( n_i \), amount of time spent computing \( t_i \), and the progress of the task \( p_i \) (in reality we can only estimate. Additionally, the state also includes the number of unassigned nodes \( u \), and the time of the world \( \tau \). The state is therefore the set \( S = \{k_1, n_1, c_1, t_1, p_1, \ldots, k_J, n_J, c_J, t_J, p_J, u, \tau\} \).

- Actions are the assignment of a node to a job which are numbered as \( A = \{1, \ldots, J\} \).

- Transition probabilities \( P \) describe the probability that a new state will be reached. \( s_{t+1} = s' \) from \( s_t = s \) when taking action \( a \). These probabilities are very difficult to calculate explicitly, but can be derived empirically with the help of simulation.

- Reward signals can be very arduous to choose. In most of the RL applications, reward is selected to be the designer’s utility, but it is known that this might not be the best reward signal. We will base our rewards and utility on the communication costs and speed of executing tasks as compared to implementing them serially on a single node. One problem with this approach is that the performance of the entire network is not known until all jobs are completed, making such evaluation quite problematic in a continuous (non-episodic) environment, although it is still possible.
5.2.2 Learning Algorithms

RL algorithms are used to find an optimal policy. If the MDP model is known, it is then possible to solve through dynamic programming algorithms, such as value iteration, to find the optimum value function. However, the transition probabilities in M are difficult to calculate (and impossible in a real-world scenario), so we must learn it, or the analogous state-value function through experience. Bread-and-butter RL techniques include Temporal Difference (TD) learning and Monte-Carlo (MC) estimation. Additionally, many non-value-function oriented approaches are available for estimating from experience including model learning, policy gradient, and many others. For this project we only make use of the on-line state-value function learning methods including because they are well suited for on-line learning [16], and we hypothesize that this problem could be solved with these simple algorithms given carefully chosen features and rewards.

Tabular TD(0) Q-learning and Sarsa These two algorithms are one-step state-value function learning algorithms for control. Q-learning is an off-policy algorithm, which means that it updates the state-action value function ($Q$) independent of the current policy with the following update rule:

$$\delta_t = r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)$$

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \delta_t$$

Where $\alpha$ is a learning rate parameter, $r_t$ is reward at time $t$, $\gamma$ reward
discount factor and \( \delta \) is the TD error. Unlike Q-learning, Sarsa is on-policy, meaning it can update \( Q \) using the policy currently being followed:

\[
\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \rightarrow Q_t(s_t, a_t) + \alpha \delta_t
\]

Both of these algorithms were implemented using hash tables to represent \( Q \). Actions were then chosen using an \( E \)-greedy policy.

**Linear Approximation Sarsa(\( \lambda \))** TD(\( \lambda \)) is a combination of Monte Carlo learning and one-step TD. Instead of performing only a single step backup like TD(0) or performing updation of all states in \( \pi \) like TD(\( \lambda \)) and Monte Carlo, it instead performs an average \( n \)-step TD backup based on the parameter \( \lambda \). This can be implemented quite efficiently on-line using an eligibility trace function to keep track of how recently states have been visited. TD(\( \lambda \)) backup can be used for on-policy control with an algorithm comparable to Sarsa called Sarsa(\( \lambda \)).

Using a tabular \( Q \) function, Sarsa(\( \lambda \)) still has all same problems with generalization as the one-step TD algorithms. To allow for greater generalization between the different states as well as the continuous state variables, \( Q \) can be represented as a continuous valued function. Simple linear function approximation and neural networks are widely used in the RL community to much success [1, 16]. We implemented a linear \( Q \) function for Sarsa(\( \lambda \)).

We are using a different function for each action. This is because it is very difficult to justify a single \( Q \) function linear in the numeric value of the action, although the indicator functions could be used. The biggest challenge in successfully applying this algorithm is extracting linear features \( F \) from a
state. For linear functions, features are often chosen to be binary values extracted from the state space through tile coding or other methods [16].

5.3 Feature Extraction and Generalization

The biggest motivation for applying RL to this problem is the possibility to create a learning agent which can manage tasks effectively in a dynamic system and handle new types of task by improving its own policy online. It is more or less futile to blindly use the algorithms described above without putting much thought into the feature extraction. This problem of generalization is quite common in RL.

To demonstrate this problem, let us suppose that a tabular Q-learning agent has learned a good $Q$ function for observations containing task numbers $1,\ldots,K$ when it is suddenly given an observation containing the $K+1$ task type. This agent will need to learn how to manage this task from scratch completely. Ideally, we would want the agent to first associate new states with the most similar state it has already experienced before and then learn from there. There are many different approaches to solve this problem. One of the possibilities is to create similarity trees over the space of state-actions [3] or aggregate similar states into bins [15]. Another option is to extend the agent’s value function by copying the verbatim to the row of the value function, which is, associated with the most similar other task type [18].

Feature extraction can be used both to reduce the state-space and to help generalize from the perspective of the agent. If we select the features by intelligently combining aspects of the underlying state, we can often turn an unmanageable problem into something which is manageable. For this project, we put some work into extraction of feature and also approach generalization through a multi-agent formulation.
5.4 Roughly Simulating a WSN Controlled by Agent

To accurately simulate a WSN, we would need to model processing nodes that are completing real computing tasks over time. This type of simulation is extremely slow. Hence, we decided to model the communication costs and processing time of theoretical tasks mathematically for our initial investigation. Such a mathematical model does not need to represent any particular computational task accurately as long as it is roughly representing the dynamics of communication/processing time trade-offs in a WSN.

Our simulator specifies an environment with several parameters: number of jobs $J$, number of nodes $N$, and number of different types of tasks $K$. The simulator then creates $K$ job types. Each job $j$ of $k$ type has $W_k$ work units that must be completed; let $k_j$ be the job type of job $j$. We use a logarithmic curve $W_k(x) = \nu_k \times \ln(x) + 1$ to calculate the amount of work $x$ nodes can complete per second for a $k$ type job; a logarithmic curve creates a model so the per node speedup decreases as more nodes are assigned to a job. Note that when no nodes are assigned to a job, no work or communication is being done and therefore $W_k(0) = C_k(0) = 0$. The coefficients $\nu_k$ and $\eta_k$ are parameters which determine the trade-off between communication and processing time as more nodes are added to a job of type $k$. In order to evaluate a WSN, we use a conversion ratio $r = 0.1$ to express a situation in which each second of computational speedup is ten times more valuable than every byte that is communicated. It is important to note that the agent does not know the value of the curves, but must learn them over period of time. Utility achieved by a WSN can then be measured after completion of every job.

5.4.1 Episodic Simplification

In our study, we simulated the WSN in an episodic manner where all jobs appear at time 0 instead of modeling a long-lived agent in a rapidly & dynamically changing WSN. This is a big simplification. Simplification decision makes the problem less sequential in nature because everything that needs to be learned is an optimal assignment of nodes at the beginning of the
episode based on the combination of task types. However, it acts as a good first step for this project.

For the episodic simulator, an agent is responsible for handling the assignment of nodes from the beginning of the episode until all jobs are completed. The number of jobs stays constant on each episode, but the type of each job is random. An agent can then take the action of keeping a node idle or assigning a node to a job. Once all the nodes are assigned to work on a job or stay idle, work is completed on a per second basis. Whenever a job is completed, all nodes assigned to the job are released.

5.4.2 Monolithic Agent

In our first experiment, we used a monolithic agent which controls the WSN as described earlier. This agent manages the nodal assignment of the entire network over an episode. Using the entire state $s \in S$ as an observation to this agent wouldn’t be feasible, so we have put some work into choosing intelligent features from this state. If we provide the agent with a subset of the state information the problem is no longer an MDP and becomes a learning problem in a partially observable MDP (POMDP). TD algorithms can provably converge in MDPs but they do not in case of POMDPs. But, in practice these algorithms do perform well with partial observation given good features. Experiments using this monolithic model have proven to be unsuccessful despite various methods of feature extraction.

5.4.3 Multi-Agent Bidding

Due to the very little chances of success of the monolithic agent, we opted to take a new approach inspired by Kumar and Shah’s work [15] optimizing non-parallel power management and computational tasks in WSNs. In their
experiment, they modeled each node as a separate Q-learning agent. Considering the parallel nature of our tasks, we modeled each separate job as an agent. Instead of controlling nodes directly, each agent has a numeric bid which it can change through actions. Nodes assign themselves to the job having the highest bid if any bid was above some threshold. Instead of waiting for all the nodes to be assigned to begin job computation, each node has the chance to assign itself once and then the computation proceeds step by step. After any node finishes its computational work or changes its assignment, each agent gets the chance to revise its bid. In the episodic case, when every job completes, all agents are rewarded with the utility earned. In the continuous case, each agent can be rewarded on its job completion. When a new job appears, a new agent is created by copying an agent which had previously been assigned to the same type of task. If this new task type has never been assigned, then we copy a random agent.

For the purpose of consistency, we will describe the new MDP which is associated with this formulation. The other agents can be thought of as part of the system dynamics which control the transition probabilities.

Recall $M = (S, A, P, R)$.

- State is as before with the addition of a current bid $b_i$ associated with each job (or agent, since there is one agent per job): $S = \{k_1, n_1, c_1, t_1, p_1, b_1, \ldots, k_J, n_J, c_J, t_J, p_J, b_J, u, \tau\}$.

- Actions are: decrease bid, increase bid, do nothing.
Advantages of this model are many:

1) The policy for each individual agent is quite simple as the agent only needs to learn how to change its bid.

2) Selection of intelligent features for a single agent is more intuitive.

3) Generalization between task types is no longer a problem since every job is one’s own agent.

Just like in the case of monolithic agent, we will introduce partial observability by extracting features in order to simplify the problem for learning agents. For this task, we will supply the agent with the following feature observations:

- Agent’s current bid
- Relation of this bid to the other bids agents’
- Whether current number of nodes is the same, less-than, or greater than the best previous number for the current combination of tasks.

This last observation is especially valuable and could possibly be used to improve the monolithic agent.
5.5 Results

We first implemented our RL algorithms on a simulated WSN with 50 Nodes. Experiments were performed with both the multi-agent bidding formulation and monolithic agent on the same problems. Figure 1 shows 6 different agents working over 50,000 episodes on the same set of $J$ tasks. The difficulty of each type $K$ task is generated randomly using the $\eta_k$ and $\nu_k$ parameters. All tasks are such that the optimal number of nodes to assign lies in the range $[1, 50]$, but each task is significantly different from the other. The agents used are as follows:

1) mon-Even is a fixed policy in which even number of nodes is assigned to each task. This is not a good policy because it involves too much of communication cost.

2) mon-One assigns a single node to each task. This is the baseline performance.

3) mon-Q is Q-learning monolithic agent.

4) mon-SarsaL is Sarsa($\lambda$) monolithic agent where the state space values are quantized into different groups.

5) bid-Sarsa is one-step Sarsa bidding agent.

6) bid-Q is Q-learning bidding agent.

As can be seen from agents mon-SarsaL, and mon-Q, the monolithic agent was not successful. This result is not that surprising. Despite some efforts at feature extraction, the state space was still too large. We hypothesize that with better choices of features, this monolithic agent concept could become more feasible. On the other hand, the node-bidding agents perform very well. Both