Optimal Actuation Strategies for Sensor/Actuator Networks

F. Thouin, R. Thommes and M.J. Coates

McGill University
Department of Electrical and Computer Engineering
3480 University, Montreal, Quebec, Canada H3A 2A7
Email: {coates,fthoui,rthomme}@ece.mcgill.ca

***Abstract*— Wireless sensor-actuator networks (SANETs), in which nodes perform actions (actuation) in response to sensor measurements and shared information, have great potential in medical and agricultural applications. In this paper, we focus on the problem of using distributed sensed data to design actuation strategies in order to elicit a desired response from the environment, whilst attempting to minimize the communication in the network. Our methodology is based on batch Q-learning; we describe a distributed approach for learning dyadic regression trees to estimate the Q-functions from collected data. Analysis and simulation indicate that substantial communication savings that can be achieved through distributed learning without significant performance deterioration. The simulations also reveal that the performance of our technique depends strongly on the amount of training data available.**

I. Introduction

Wireless sensor and actuator networks (SANETs) represent an important extension of sensor networks, allowing nodes within the network to make autonomous decisions and perform actions (actuation) in response to sensor measurements and shared information. The potential applications of such wireless SANETs are widespread, including agricultural maintenance, precision farming and automatic irrigation [1], [2], as well as localized delivery of medication. One of the important tasks in a SANET is the fusion of the sensed data in order to learn the response of the environment to actuations; this permits the subsequent design of an optimal actuation strategy.

A dynamic actuation strategy is a set of decision rules, one per actuation interval (or time-step), that determine the nature of the actuation performed by the SANET. It forms a map from a set of past and present measured environmental variables (and potentially the actuation history) to a discrete actuation; an example from the agricultural setting is the decision of how much herbicide and water to administer based on recent measurements of temperature, soil salinity and moisture, and weed density.

In this paper, we focus on the problem of using distributed sensor measurements to design actuation strategies in order to elicit a desired response from the environment. We are interested in the particular case where system dynamics are unknown but a batch of data is available from previous actuation epochs; our goal is to use the data to learn the system dynamics and to estimate an optimal actuation strategy, i.e, a strategy that achieves the best (expected) marginal response from the system over the course of an actuation epoch.

Our methodology incorporates distributed dyadic regression tree learning [3] into a batch Q-learning framework [4], [5]. A centralized implementation of the methodology requires significant communication, particularly when the data dimension and volume are large. For this reason, we propose two algorithms for performing distributed construction of the dyadic regression tree.

The paper is structured as follows. Section II defines the problem and states our objective. Section III provides a review of batch Q-learning and dyadic regression tree learning, and outlines our proposed methodology for merging these techniques to develop actuation strategies. Section IV describes a distributed implementation and explores communication requirements. Section V discusses the results of simulations conducted to explore the performance of our proposed algorithms.

A. Related Work and Contribution

Our work is founded upon the Q-learning framework [6] and more specifically on batch Q-learning (or A-learning), as outlined in [5], [7]–[9]. Within this framework, we incorporate functional approximation, an approach which has been adopted in temporal learning and dynamic programming for many years (see [4], [10]–[12] and the references therein). In particular, we employ a dyadic regression tree approach [3] and describe distributed implementations of this algorithm (both exact and approximate). The exact algorithm involves the identification and sharing of sufficient statistics, and is similar to the approach adopted in [13]. The approximate algorithm generates a tree that is similar but not exactly the same as that constructed by the centralized algorithm. This second algorithm is closer in spirit to the approach in [14] for the derivation of an approximate decision tree based on the transmission of a selected subset of data, and related to the classification technique of bagging using disjoint data sets as discussed in [15].

Our contributions in solving two problems related to wireless SANETs are the following. First, we propose a novel methodology for the design of an actuation strategy that maximizes the expected response from the unknown environment monitored and controlled by the SANET. Second, we address the issue of minimizing the communication costs for a low-power wireless SANET by developing two distributed algorithms to build regression trees. Both algorithms share the computational load among all nodes, which allows them to transmit a reduced set of statistics to the fusion center.

II. Problem Statement

We consider a wireless SANET consisting of joint sensor-actuator nodes labeled $i = 1, \dots, K$. Measurements are made over an epoch of T discrete time intervals. At the beginning of each interval $t = 0, \dots, T$, node i measures a set of environmental variables $V_t^{(i)}$, and chooses an actuation $A_t^{(i)}$ belonging to the discrete set of actuations $\mathcal{A}$. At the end of the epoch, each node measures a local response variable $Y_T^{(i)}$. We denote the observed data at the end of the epoch as $\{X_{0:T}, A_T, Y_T\}$, where $X_{0:t} = \{A_{0:t-1}^{(i)}, V_{0:t}^{(i)}\}_{i=1}^{K}$.

Define a dynamic actuation strategy as a collection of functions $\pi_t(X_{0:t})$ that at each time t generate an actuation A_t based on the observed data and actuation history $X_{0:t}$. We represent this dynamic actuation strategy as a vector $\pi = (\pi_0, \dots, \pi_T)$. Denote the class of actuation strategies that have non-zero probability of appearing in the observed data as D_P, where P is the distribution of the observed data. Our objective is to identify the dynamic actuation strategy $\pi^* \in D_P$ that maximizes the marginal mean response $E_P[Y_T|\pi]$.

We assume that measurements at different nodes are independent. We do not assume stationarity or that the system dynamics are Markovian (although our simulation results in Section V address a Markovian system). Finally, we assume that there are no *unmeasured direct confounders* in the observed data [7]. This implies that the actuation decision A_t is independent of *potential* outcomes given the actuation and measurement history $\{X_{0:t}\}$, i.e., no unmeasured environmental variable can have a direct causal influence on both the actuation decision and the future outcome.

III. Methodology

A. Batch Q-learning

Our approach is founded on batch Q-learning, as discussed in [4]–[6], [9]. The word "batch" indicates that learning occurs after the collection of a set of training data (possibly from multiple epochs). In reviewing Q-learning, we summarize the description in [5] for the special case where there are no intermediate rewards. The *value function* for an actuation strategy π and initial observation v_0 is defined as:

$$J_\pi(v_0) = E_\pi\left[Y_T | V_0 = v_0\right].$$

The *t-value function* at time t is

$$J_{\pi,t}(x_{0:t}) = E_\pi\left[Y_T | X_{0:t} = x_{0:t}\right].$$

The *Q-function* at time t is

$$Q_t(x_{0:t}, a_t) = E\left[Y_T | X_{0:t} = x_{0:t}, A_t = a_t\right].$$

Note that this last expectation is independent of the policy π,

The optimal value function $J^*(v_0)$ is

$$J_0^*(v_0) = \max_{\pi \in D_p} J_\pi(v_0),$$

and the optimal t-value function for history $x_{0:t}$ is

$$J_t^*(x_{0:t}) = \max_{\pi \in D_p} J_{\pi,t}(x_{0:t}).$$

For $t = T$, the optimal value function is:

$$J_T^*(x_{0:t}) = \max_{a_T \in \mathcal{A}} Q_T(x_{0:T}, a_T).$$

For $t < T$, the optimal value functions satisfy the Bellman equations [16]:

$$\begin{aligned} J_t^*(x_{0:t}) &= \max_{a_t \in \mathcal{A}} E\left[J_{t+1}^*(X_{0:t+1}) | X_{0:t} = x_{0:t}, A_t = a_t\right] \\ &= \max_{a_t \in \mathcal{A}} Q_t^*(x_{0:t}, a_t), \end{aligned}$$

where $Q_t^*(x_{0:t}, a_t)$ is the optimal time-t Q-function:

$$Q_t^*(x_{0:t}, a_t) = E\left[J_{t+1}^*(X_{0:t+1}) | X_{0:t} = x_{0:t}, A_t = a_t\right].$$

The optimal strategy π^* must satisfy, for $t \in 0, \dots, T$,

$$\pi_t^*(x_{0:t}) = \arg\max_{a_t \in \mathcal{A}} \left[J_{t+1}^*(X_{0:t+1}) | X_{0:t} = x_{0:t}, A_t = a_t\right].$$

When the distributions are unknown, the Q-functions must be estimated from the data, using an approximator such as a neural network or decision tree [4], [5], [12]. In this paper, we employ a dyadic regression tree. The estimated time-t Q-function is denoted $\widehat{Q_t}(x_{0:t}, a_t)$. The batch learning procedure thus initially involves the formation of $\widehat{Q_T}(x_{0:T}, a_T)$ based on the data $\{x_{0:T}, a_T, y_T\}$. Subsequently we set

$$\widehat{J_T^*}(x_{0:T}) = \max_{a_T \in \mathcal{A}} \widehat{Q_T}(x_{0:T}, a_T).$$

Now we can construct the data set $\{x_{0:T-1}, a_{T-1}, \widehat{J_T^*}(x_{0:T})\}$ and use it to form an estimate $\widehat{Q_{T-1}^*}(x_{0:T-1}, a_{T-1})$ of the time $T-1$ optimal Q-function. This process is repeated until we reach $t = 0$, at which point we have formed estimates for all J_t^* and hence identified our estimated optimal strategy $\widehat{\pi^*}$.

Although there has been concern about the convergence properties when incorporating function estimation in the Q-learning framework [4], [10]–[12], Murphy provides a finite sample upper bound on the generalization error of batch Q-learning in [5]. Theorem 1 therein indicates that if a suitably powerful regression approach is adopted and the complexity (or smoothness) of the underlying Q-function satisfies certain constraints, then batch Q-learning as described above is a probably approximate correct (PAC) reinforcement learning algorithm as defined by Fiechter [17]. The dyadic regression tree approach that we have chosen is near-optimal if the underlying function satisfies certain smoothness properties (see [18]); we conjecture that this near-optimality is sufficient to satisfy the constraints of Theorem 1 in [5], but we are yet to develop a proof.

B. Dyadic Regression Tree Estimation

In this section we describe our estimator and the mechanism for its computation. We address the situation where the number of potential actuations is small; this leads us to an approach where we construct a different estimator for each actuation. If the number of actuations is large, then we must assume some form of smooth behaviour of Q-functions with respect to actuation and incorporate the actuation level as one of the regression parameters. We now describe the process for forming an estimate $\widehat{Q_T^*}(x_{0:T}, a_T)$ based on the response measurements $\{x_{0:T}, a_T, y_T\}$. Subsequent to this process, we form estimates $\widehat{Q_t^*}(x_{0:t}, a_t)$ in exactly the same fashion based on the constructed data sets $\{x_{0:t}, a_t, \widehat{J_t^*}(x_{0:t+1})\}$.

We model a response measurement Y_T to a specific actuation a_T as the expected response plus additive Gaussian noise.

$$Y_T(s_T, a_T) = Q_T(s_T, a_T) + \epsilon_T \ . \tag{1}$$

Here s_T is the set of variables (or sufficient statistics) on which Y_T (Q_T) directly depends, aside from the actuation a_T. For example, in a Markov system, $S_T = V_T$. The term $\epsilon_T \sim \mathcal{N}(0, \sigma^2)$ denotes zero-mean Gaussian with variance σ^2.

The expected response $Q_T(S_T, a_T)$ is modelled as a piecewise smooth function, and we use a complexity-penalized dyadic regression tree procedure to formulate an estimate of the function, based on the approach described in [3]. We form a dyadic partitioning of the S_T space. The partitioning is achieved through the construction of a tree. The root represents a cell encompassing the entire space. At each layer in the tree we divide the cell in half in one of the dimensions; as we progress through subsequent layers, we cycle through the dimensions in turn. The resultant data structure is a tree of depth $d_{\max}$, with a small, possibly empty, subset of the samples $S_T^{(i)}$ residing at any given leaf-node.

At each node in the tree we fit a function to the descendant data points using least squares. This function could be a wedgelet or a low-order polynomial, but in this paper, we adopt the computationally simpler estimate of a constant (the mean) over each cell in the partition. This choice greatly simplifies the distributed construction of a tree in the case that all data are not transmitted to a fusion site. We associate with each node in the tree a loss (or risk) value which is the sum of squared errors for the chosen fitted function. This is appropriate for the Gaussian model we have adopted.

The formulation of the final estimate is achieved by pruning the tree using a complexity penalty. The complexity penalty is proportional to the square-root of the number of leaves in a candidate tree. Through a recursive procedure, we identify, for each value M, the M-leaf tree that has minimum risk (squared error). From this set of trees, we select as our final regression estimate the tree that has minimum penalized risk (the sum of the risk and the complexity penalty). The choice of the complexity penalty, which is related to the number of data points and the noise variance σ^2, is discussed in more detail in [3]. Note that in our case, the variance σ^2 is unknown, so the complexity penalty becomes in essence a tuning parameter that controls the smoothness of our estimates.

IV. Distributed regression trees

In this section we consider the problem of creating a regression tree from data measurements spread across multiple SANET nodes. For concreteness, we focus on the formation of the first regression tree in the Q-learning algorithm. Specifically, we examine the case where K nodes (or clusterheads) each measure (or collect) N data sets of the form $\{S_T, A_T, Y_T\}$ as defined in Section II. Denote by R the dimension of S_T. Each sensor node is assigned a unique identification number $i \in 1, \ldots, K$, and node K acts as a fusion center. Let $s_{i,m}$ represent the m-th data set recorded by the i-th node.

We consider that a communication tree has been formed to allow each node to relay data to the fusion node K. We denote the number of hops from node i to the fusion node as $G(i)$, and the average number of hops as $\bar{G}$. Denote, for each node i, a set of descendants $\mathcal{D}(i)$ and a set of children $\mathcal{P}(i)$. Let $\mathcal{D}^+(i)$ be the set $\{\mathcal{D}(i), i\}$ containing node i and all of its descendants. In our analysis of communication overhead, we assume perfect communications, disregarding the possibility of node failures, and ignore packet overhead bits. We now outline three techniques used to build a regression tree from the distributed data.

A. Algorithm Descriptions

1) Uncompressed Data Aggregation (UDA): Each node i sends its N measurement vectors to its parent node until node K has all KN measurement vectors. Node K then performs all the operations necessary to identify the regression tree.

2) Compressed Data Aggregation (CDA): This algorithm is a special case of the technique for distributed learning described in [13]. The transmitted data is reduced by requiring each SANET node to determine and aggregate sufficient statistics for the creation of the regression tree. The information ultimately received by node K allows it to deduce exactly the same tree as in the UDA case. Initially, each node grows a decision tree to depth $d_{\max}$ based on its N data points. Each node i then calculates the mean value $\widehat{\mu}_{i,j}$ and sum of squares error $\widehat{\sigma}^2_{i,j}$ for each leaf cell j. The number of node i's data points falling within leaf cell j is denoted by $n_{i,j}$.

The procedure to transmit the statistics to node K is the following: each node i receives three vectors ($\widehat{\mu}$, $\widehat{\sigma}^2$ and n) from each of its children, calculates the aggregated statistics with equations (2)-(4) below for each non-empty cell, and then transmits these to its parent in the communication tree. When node K has received all the data, it incorporates its own and then constructs the regression tree with sufficient statistics n'_j, $\widehat{\mu}'_j$ and $\widehat{\sigma}'^2_j$.

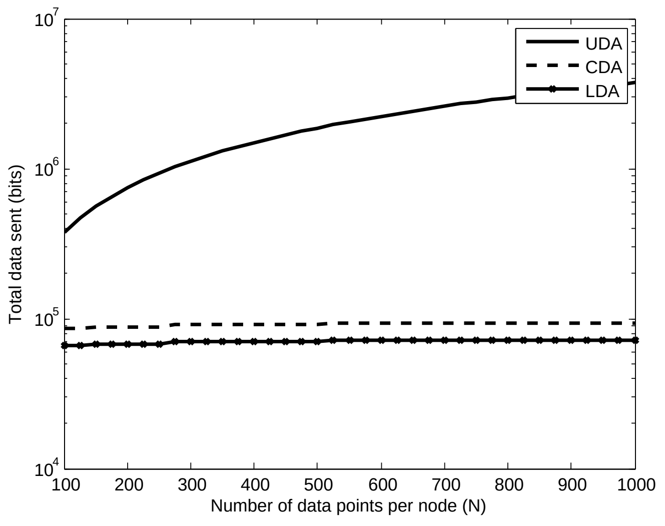


Fig. 1. Total bits transmitted as a function of the number of data points N at each of the $K = 32$ nodes. $\alpha = 0.95$ and $\gamma = 1$.

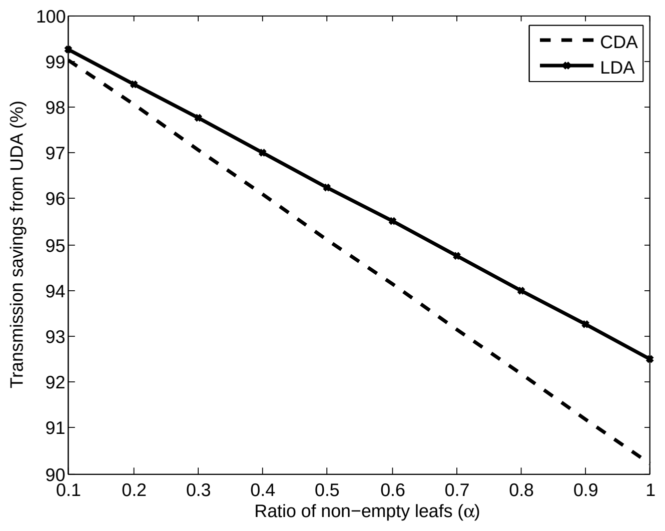


Fig. 2. Transmission savings in % of CDA and LDA over UDA as a function of the fraction of non-empty leaf cells α. For this simulation, we set $N = 256$, $b_2 = 13$, $\gamma = 1$ and $\overline{G} = 2.5$.

$$n'_{i,j} \leftarrow n_{i,j} + \sum_{k \in \mathcal{P}(i)} n'_{k,j} \tag{2}$$

$$\widehat{\mu'}_{i,j} \leftarrow \frac{n_{i,j} \cdot \widehat{\mu}_{i,j} + \sum_{k \in \mathcal{P}(i)} n'_{k,j} \cdot \widehat{\mu'}_{k,j}}{n'_{i,j}} \tag{3}$$

$$\begin{aligned} \widehat{\sigma'}^2_{i,j} \leftarrow & \sum_{k \in \mathcal{P}(i)} \widehat{\sigma'}^2_{k,j} + n'_{k,j}(\widehat{\mu'}_{k,j} - \widehat{\mu'}_{i,j})^2 \\ & + \sum_{m: s_{i,m} \in j} (y_{i,m} - \widehat{\mu'}_{i,j})^2 \end{aligned} \tag{4}$$

3) Lossy Data Aggregation (LDA): LDA further decreases the amount of data that is sent, but derives a regression tree which may differ from that created by UDA and CDA. The LDA algorithm consists of three stages. In the first stage, each node grows an individual tree based on its own measurements, and transmits the indices of the leaf cells in its tree via the communication tree to the fusion node K. In the second stage, node K calculates the number of instances t_j that cell j appears in the trees. At this point, node K applies a heuristic to decide on the unified tree. Each value of t_j must exceed a threshold t_{min} for cell j to be included in the tree. A natural choice is $t_{min} = K/2$: at least half of the sensor nodes must have elected to include a given cell in their individual tree in order for node K to include the cell in the unified tree. Node K then transmits the list of leaf cells in the unified tree. In the third stage of the algorithm, each of the other nodes calculates its mean value $\widehat{\mu}_{i,j}$ and number of points $n_{i,j}$ for each cell j in the unified tree. These are relayed back to the fusion node using the same aggregation procedure described in the CDA algorithm.

B. Communication Costs

In this section we perform an analysis of the amount of data that is sent under each algorithm. We assume that a b_1-bit representation is used for each mean $\widehat{\mu'}_{i,j}$, sum-of-squares error $\widehat{\sigma'}^2_{i,j}$, and each component of data vectors $\{S_T, Y_T\}$. The number of points falling in a cell, $n_{i,j}$, is represented by a b_2-bit integer.

1) UDA: Node i has to send its N measurements, each of which contains $R + 1$ components, in addition to those it is forwarding from its children. The total number of bits sent throughout the network, H_{UDA}, is:

$$H_{UDA} = \bar{G}(K-1)N(R+1)b_1 \tag{5}$$

2) CDA: Each node i sends the values of $\widehat{\mu'}_{i,j}$, $\widehat{\sigma'}^2_{i,j}$, and $n_{i,j}$ for all non-empty leaf cells in the set of unpruned trees constructed by the set of SANET nodes $\mathcal{D}^+(i)$. Denote this number of non-empty cells by $C(i)$ and the average number by $\bar{C}$. Node i must also send some overhead bits that index the set of cells to which the statistics correspond. As discussed in [19], we can encode the structure of a dyadic tree with M leaf cells using at most $2M$ indexing bits. The total number of bits sent throughout the network, H_{CDA}, is:

$$\begin{aligned} H_{CDA} &= (2b_1 + b_2 + 2) \sum_{i=1}^{K-1} C(i) & (6) \\ &= (2b_1 + b_2 + 2)(K-1)\bar{C} & (7) \end{aligned}$$

3) LDA: Denote by $M(i)$ the number of leaf cells in the pruned tree at node i; let $\bar{M}$ be the average number across all $K - 1$ nodes. In the first stage of LDA, each node i sends a vector of $2M(i)$ bits over G_i hops to indicate to node K the structure of its reduced tree of $M(i)$ leaf cells. In the second stage, node K decides on the structure of the unified tree and sends the identities of its L leaf cells to the other $K - 1$ nodes. The communication cost is simply the indexing overhead $2L$ for each of the $K - 1$ nodes. Finally, the same aggregation step is performed as in CDA, except that the tree is common to all nodes, and the sum-of-squares error is not

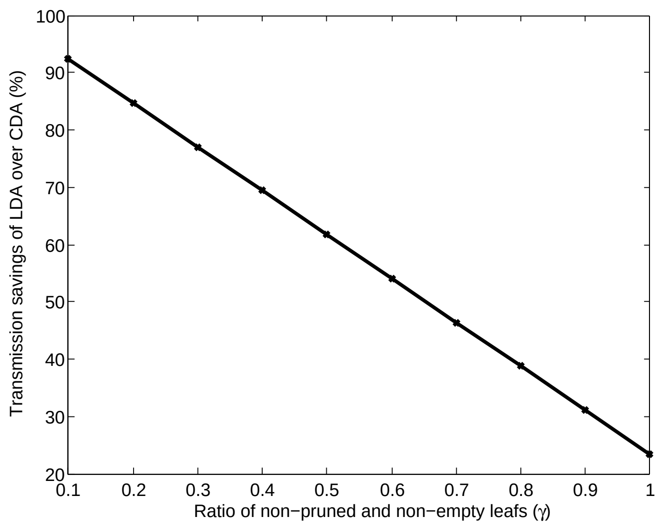


Fig. 3. Transmission savings in % of LDA over CDA as a function of the fraction of non-pruned/non-empty leaf cells γ. For $\bar{G} = 8$, we have $b_1 = 2\bar{G}$ so $H_{LDA}/H_{CDA} = \gamma$. For this simulation, we set $N = 256$, $b_2 = 13$ and $\alpha = 0.95$.

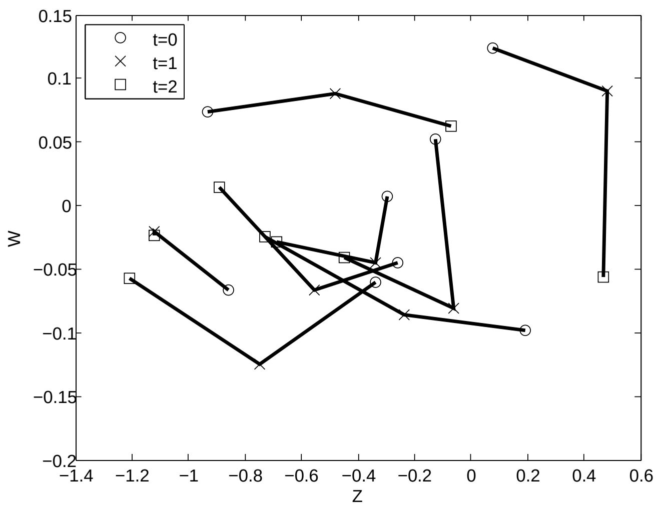


Fig. 4. Values of W and Z generated with the update rules for eight pairs (W_0, Z_0) during two iterations (T=2).

transmitted. There is no need for additional indexing bits in this final stage because the set of leaf cells is already known. The total transmission cost, H_{LDA}, is:

$$H_{LDA} = (K-1)L(b_1 + b_2 + 2) + 2\sum_{i=1}^{K-1} G_i M(i) \tag{8}$$

C. Comparative Analysis

In order to more readily compare the communication overhead of the three algorithms, we introduce some extra notation and make some simplifying assumptions. Denote by α the average fraction of non-empty leaf cells in the CDA trees, i.e.,

$$\alpha = \frac{\bar{C}}{2^{d_{\max}-1}}.$$

Denote by γ the ratio of L and $\bar{C}$. Note that when $t_{\min} = K/2$, we have $L < 2\bar{C}$ and hence $\gamma < 2$ (consider the extreme case where each of the L leaf cells is included in the trees of exactly $K/2+1$ nodes and there are no other leaf cells in any tree). In practice, we expect that in most cases $\gamma < 1$ due to the two instances of pruning in LDA, first at the individual nodes and second during the construction of the unified tree. We assume that G and M are independent, so that the summation in (8) can be approximated as $(K-1)\bar{G}\bar{M}$, and we further assume that $L \approx \bar{M}$ is a reasonable approximation.

These assumptions lead to the following relationships:

$$H_{CDA}/H_{UDA} = \frac{\alpha 2^{d_{\max}-1}(2b_1 + b_2 + 2)}{\bar{G}Nb_1(R+1)} \tag{9}$$

$$H_{LDA}/H_{CDA} = \frac{\gamma(b_1 + b_2 + 2\bar{G} + 2)}{2b_1 + b_2 + 2} \tag{10}$$

Example: We consider a network with $K = 32$ nodes where the average number of hops $\bar{G} = 2.5$ in the communication tree. Each node gathers N data points for each of $R = 2$ variables. For CDA and LDA, we use $d_{\max} = 7$. Finally, we set $b_1 = 16$ and $b_2 = \log_2(NK)$.

Fig. 1 illustrates the number of bits transmitted by each algorithm as a function of the number of data points, under the assumptions we have outlined. We observe how the number of bits sent by UDA grows with the number of data points gathered at each node whereas the two algorithms we propose are independent of N. For $N > 300$, the communication cost for UDA is one order of magnitude more than CDA and LDA.

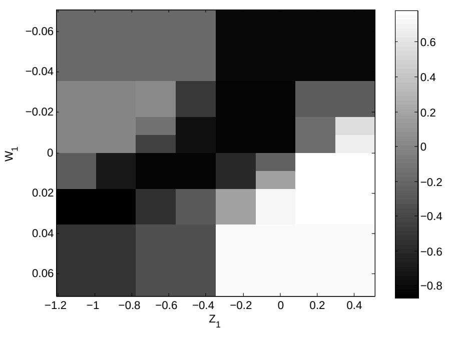
We examine the communication performance of CDA and LDA with (11) by calculating the percentage reduction with respect to UDA:

$$\textbf{Savings}(H) = \frac{H_{UDA} - H}{H_{UDA}} \times 100\% \tag{11}$$

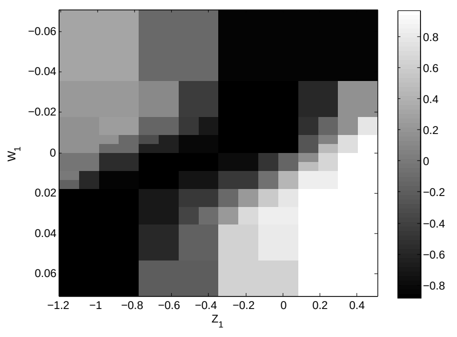
Fig. 2 shows the savings achieved by CDA and LDA as the value of α varies for the case $\gamma = 1$. This is the scenario in which the unified tree has $\bar{C}$ leaf cells. The plot indicates that when the percentage of occupied leaf cells (α) is small, the communication reduction is significant (99 percent in this case) and both LDA and CDA achieve similar savings. The savings diminish as α increases, reducing to 90 percent for CDA in the case where all cells are occupied ($\alpha = 1$). The difference between CDA and LDA increases as the pruning becomes more substantial (the unified tree is more compact); the extent of pruning is indicated by the parameter γ and shown in Fig. 3.

V. Simulation experiment

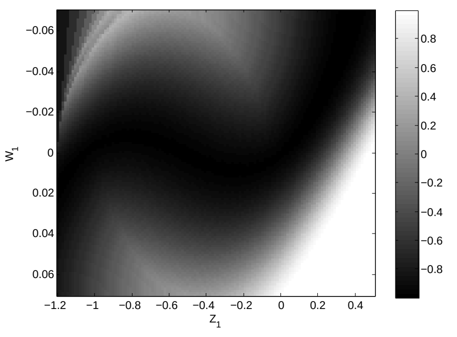
We perform simulations to illustrate the performance of the proposed estimation method and the LDA algorithm for a fictitious example scenario. We consider a model of an agricultural system, in which actuation corresponds to the release of a herbicide. Each wireless SANET node measures the soil moisture content with Z_t and W_t and at the start of a day t and makes a decision A_t (-1 or 1) about the release of



(a) $\widehat{J^*}_1$(LDA)

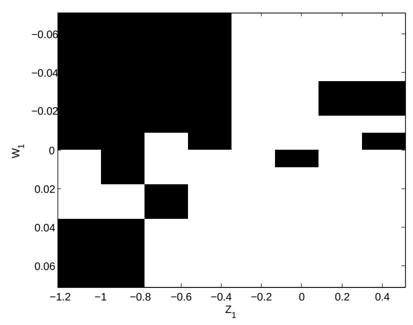


(b) $\widehat{J^*}_1$(UDA)

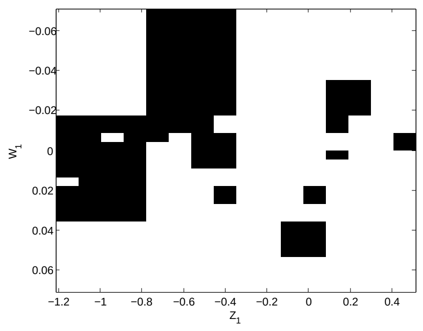


(c) J_1^*

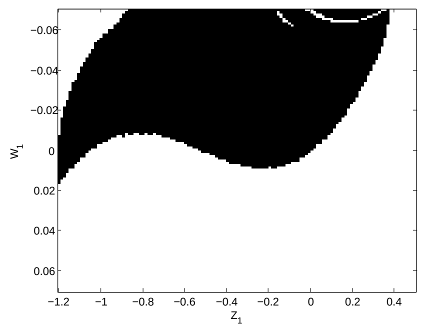
Fig. 5. Optimal J function, J_1^*, and the two estimated functions, $\widehat{J_1^*}$(UDA) and $\widehat{J_1^*}$(LDA) for the dynamic model described in Section V.Training is performed with 8192 data points and noise strength $\sigma = 0.2$.



(a) $\widehat{\pi^*}_1$(LDA)



(b) $\widehat{\pi^*}_1$(UDA)



(c) π_1^*

Fig. 6. Comparison between optimal strategy π_1^* and the estimated strategies $\widehat{\pi_1^*}$(UDA) and $\widehat{\pi_1^*}$(LDA). The white areas represent $a_1 = 1$ whereas the black areas represent $a_1 = -1$. Training is performed with 8192 data points and noise strength $\sigma = 0.2$.

the herbicide. At the end of the epoch, it measures the change in the ratio of plant-density to weed-density Y_T, estimating it from digital images of the surrounding environment. We wish to emphasize that the dynamic model below is merely chosen to illustrate the behaviour of the learning algorithm; we are not suggesting that this model bears any resemblance to an appropriate model for the agricultural system.

The model of the system is described by the following equations (these are used to generate the measurements in our simulations). Initially, we have

$$Z_0 \sim U[-0.07, 0.07] + 0.05\epsilon \tag{12}$$

$$W_0 \sim U[-1.2, 0.5] + 0.05\epsilon \tag{13}$$

where ϵ is zero-mean Gaussian noise with variance 1.

For $t > 0$, at each of the T iterations in one epoch, the following update is performed:

```
1: $W'_0 = W_{t-1}$
2: $Z'_0 = Z_{t-1}$
3: for $i = 1 : 6$ do
4:     $W'_i = W'_{i-1} + 0.001A_{t-1} - 0.0075\cos(3Z'_{i-1})$
5:     $Z'_i = Z'_{i-1} + W'_i$
6: end for
7: $W_t = W'_i$
8: $Z_t = Z'_i$
```

where W is bounded in $[-1.2, 0.5]$ and Z in $[-0.07, 0.07]$. If W'_i lies outside the range after the update, it is set to the boundary value. If Z'_i lies outside its range, it is set to the boundary value as well and W'_i is set to 0.

After the update, at the end of the iteration, noise is added to the measurements:

$$W_t = W_t + 0.05\epsilon \tag{14}$$

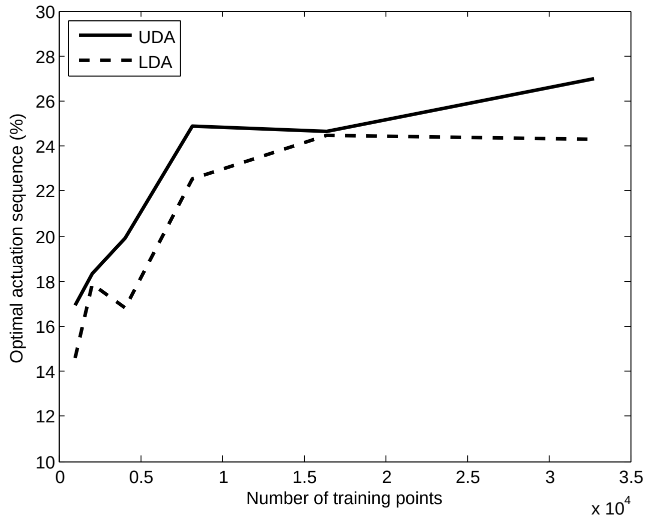
$$Z_t = Z_t + 0.05\epsilon \tag{15}$$

At the end of the epoch, the local response Y_T is generated with the following expression:

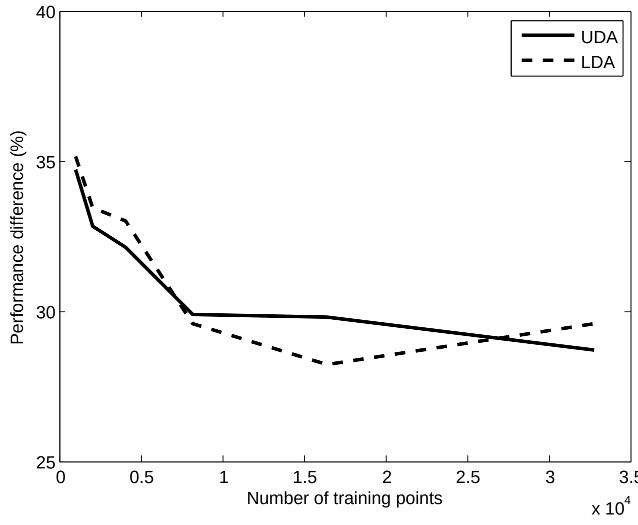
$$Y_T = \sin(3Z_T) + \sigma\epsilon \tag{16}$$

where σ^2 represents the noise variance on the measured response.

The results we report were generated by averaging over 50 simulations with different training data sets (S, V, Y) for each setting of the parameters we vary. In order to generate the training data, actuation decisions were made at random with equal probability. All results concern the case where $T = 2$; Fig. 4, shows an example of W and Z generated with these update rules. In the execution of our algorithms, we adopted a Markov model, i.e., we set $S_t = V_t$.



(a) Ratio of optimal actuation sequences



(b) Relative performance difference

Fig. 7. (a) The percentage of actuation sequences that match the optimal sequence ($\widehat{a}_{0:T} = a^*_{0:T}$)as a function of the training set size. (b) The relative performance difference (as expressed in (17)) as a function of the training set size. The distributed algorithm used is LDA with number of nodes $K = 8$. Noise strength coefficient $\sigma = 0.2$ and epoch duration $T = 3$. Values are the average results obtained from 50 training sets tested with 80,000 points.

We first compare the performance of our centralized learning algorithm (UDA) to the distributed LDA algorithm. We assume a communication hierarchy where 8 clusterheads first gather data from neighbouring SANET nodes and then execute the LDA algorithm.

Fig. 5 shows the optimal J function, J_1^*, and the two estimated functions, $\widehat{J^*}_1$(UDA) and $\widehat{J^*}_1$(LDA) for a typical simulation. The dyadic regression tree has difficulty matching the highly non-linear structure of the J function, but the generated estimates are reasonable. The estimate generated by the LDA algorithm is slightly less accurate than the centralized version. Fig. 6 compares the optimal strategy π_1^* with $\widehat{\pi^*}_1$(UDA) and $\widehat{\pi^*}_1$(LDA) for the same simulation. In this example, the difference in the responses evoked by $a_1 = 1$ and $a_1 = -1$ is relatively small for much of the W-Z plane, so small errors in approximating $Q_1(a_0)$ and $Q_1(a_1)$ can lead to suboptimal actuation decisions.

We are interested in examining the impact of the number of training points (NK) on the performance of the algorithms; We ran simulations for six different training set sizes, ranging from 1024 to 32768. Fig. 7(a) shows the percentage of correctly identified actuation strategies for the two algorithms. Fig. 7(b) displays the performance difference for the centralized and distributed approach as expressed in (17). In each case, these values are estimated by comparing the optimal and estimated actuation strategies of 80,000 randomly generated data sets.

$$\mathbf{PD} = E\left[\frac{J^*(v_0) - \widehat{J^*}(v_0)}{J^*(v_0)}\right] \cdot 100\% \qquad (17)$$

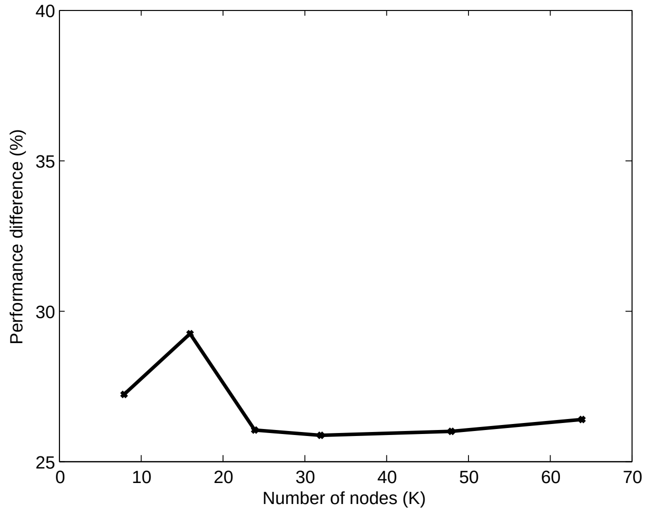
We observe that as the size of the training set increases, the percentage of correctly-identified actuation sequences increases and the performance difference decreases. Both figures indicate that the performance of the LDA algorithm is almost as good as UDA for all training set sizes.

In Fig. 8(a) we show the impact of the number of nodes K under LDA on the performance difference PD for a fixed number of training points $NK = 8192$. From the plot, we observe that increasing the number of nodes up to $K = 32$ actually yields better results. This can be explained by the fact that by dividing the data and only keeping the leaf cells that appear in at least half of the nodes, bad data gets averaged and creates less disturbance on the estimate. However, for $K > 32$, the PD starts to increase again because the number of points available at each node becomes insufficient to construct an accurate tree.

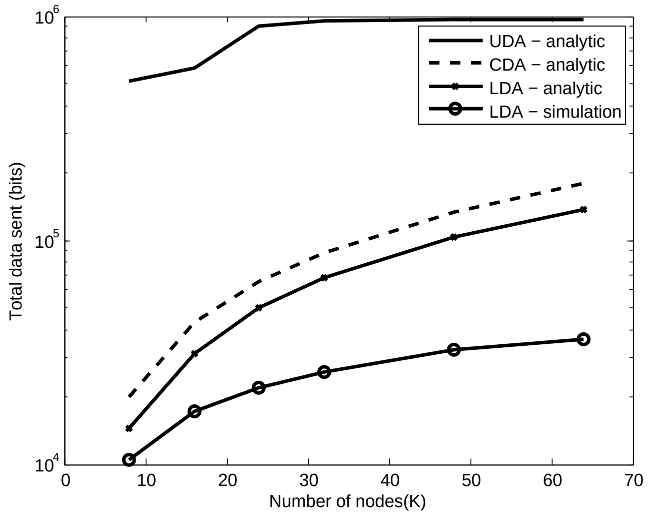
Fig. 8(b) depicts the calculated (UDA, CDA and LDA) and simulated (LDA) number of bits transmitted during the learning procedure. The analytical curves were generated by combining (5), (6) and (8) with the approximations proposed in Section IV-C. From the figure, we observe that the number of bits sent under LDA during the simulations is lower than our calculated cost. Contrary to our approximation $L \approx \bar{M}$, the number of leaf cells L in the unified tree is typically much smaller than $\bar{M}$ during the simulations, which leads to less communications.

VI. Conclusion

In this paper, we addressed the problem of determining an actuation strategy that maximizes the marginal mean response for a wireless SANET. Our approach incorporates dyadic regression tree estimation into a batch Q-learning. In the future, we will investigate the development of adaptive training strategies over multiple epochs, focusing on how to determine regions of the input space that need more training. We will also pursue other estimation strategies, such as trees generated by



(a) Relative performance difference



(b) Total bits transmitted

Fig. 8. (a) The relative performance difference of LDA (as expressed in (17)) as a function of the number of nodes K. (b) Total bits sent during one iteration as a function of the number of nodes K. We show values for UDA, calculated upper-bounds for CDA and LDA, and simulated values for LDA. Total number of data points used for both training and testing is $NK = 8192$, noise variance $\sigma^2 = 0.04$ and $T = 3$. Simulated values are the average results obtained from 50 training sets tested with 80,000 data points.

uneven splits which allow more accurate approximation, for example cycle-spinning tree techniques [20].

A centralized implementation involves a substantial communication overhead. We presented two algorithms for distributed tree learning in order to reduce the transmission cost. Our first proposal reduces the amount of data sent through the identification and sharing of sufficient statistics, and it allows the construction of the same tree as in the centralized case. The second scheme further reduces the communication overhead, but potentially deduces a tree different from that identified by the centralized algorithm. Based on analytical calculations and simulations, we observe that our second technique shows important communications savings over the centralized approach (more than 90%) whilst maintaining similar performance. Performance is dependent on the total number of training data points; increasing the number of nodes while reducing the number of measurements at each node does not negatively impact performance. In the future, we will develop an algorithm based on the construction of forests of trees.

References

[1] N. Wang, N. Zhang, and M. Wang, "Wireless sensors in agriculture and food industry—recent development and future perspective," *Computers and Electronics in Agriculture*, vol. 50, pp. 1–14, Jan. 2006.

[2] R. Evans and J. Bergman, "Relationships between cropping sequences and irrigation frequency under self-propelled irrigation systems in the Northern Great Plains (NGP)," USDA, Tech. Rep., 2003, project Number: 5436-13210-003-02.

[3] C. Scott, R. Willett, and R. Nowak, "CORT: Classification or regression trees," in *Proc. IEEE ICASSP*, Hong Kong, China, April 2003.

[4] L. Kaebling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *J. Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[5] S. A. Murphy, "A generalization error for Q-Learning," *J. Mach. Learn. Res.*, vol. 6, pp. 1073–1097, Sept. 2005.

[6] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[7] S. Murphy, "Optimal dynamic treatment regimes," *J. Roy. Stat. Soc., Ser. B*, vol. 65, no. 2, pp. 331–366, 2003.

[8] J. Robins, "Discussion of "Optimal dyamic treatment regimes" by S.A. Murphy," *J. Roy. Stat. Soc., Series B*, vol. 65, no. 2, pp. 355–356, 2003.

[9] D. Blatt, S. Murphy, and J. Zhu, "A-learning for approximate planning," The Methodology Center, Penn. State University, Tech. Rep., 2004.

[10] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proc. 1993 Connectionist Models Summer School*, M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1993, pp. 255–263.

[11] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Automatic Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[12] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[13] D. Caragea, A. Silvescu, and V. Honovar, "A framework for learning from distributed data using sufficient statistics and its application to learning decision trees," *Int. J. Hybrid Intelligent Systems*, vol. 1, no. 2, pp. 80–89, 2004.

[14] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff, "Hierarchical decision tree induction in distributed genomic databases," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 8, pp. 1138–1151, 2005.

[15] N. Chawla, T. Moore, K. Bowyer, L. Hall, C. Springer, and W. Kegelmeyer, "Bagging is a small dataset phenomenon," in *Proc. of Conf. of Computer Vision and Pattern Recognition (CVPR)*, Hilton Head Island, SC, June 2000.

[16] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[17] C. Fiechter, "Expected mistake bound model for on-line reinforcement learning," in *Proc. Int. Conf. Machine Learning*, Nashville, TN, July 1997, pp. 116–124.

[18] E. Kolaczyk and R. Nowak, "Multiscale likelihood analysis and complexity penalized estimation," *Annals of Statistics*, vol. 32, pp. 500–527, 2004.

[19] A. Cohen, I. Daubechies, O. Guleryuz, and M. Orchard, "On the importance of combining wavelet-based nonlinear approximation with coding strategies," *IEEE Trans. Inform. Theory*, vol. 48, no. 7, pp. 1895–1921, 2002.

[20] R. R. Coifman and D. L. Donoho, "Translation-invariant de-noising," Stanford University, Department of Statistics, Tech. Rep., 1995.