Reinforcement Learning for Link Adaptation in MIMO-OFDM Wireless Systems

Sungho Yun
Wireless Networking and Communications Group
Electrical and Computer Engineering
the University of Texas at Austin
Austin, Texas 78712–0240
Email: shyun@mail.utexas.edu

Constantine Caramanis
Wireless Networking and Communications Group
Electrical and Computer Engineering
the University of Texas at Austin
Austin, Texas 78712–0240
Email: caramanis@mail.utexas.edu

***Abstract*—Machine learning algorithms have recently attracted much interest for effective link adaptation due to their flexibility and ability to capture more environmental effects implicitly than classical adaptation algorithms. However, past applications are limited to rather simple configurations such as identifying channel condition or link adaptation in fixed or slowly varying channels. Recently, more sophisticated approaches using offline supervised learning have been proposed for link adaptation in complex configurations such as MIMO-OFDM. However, their time complexity and offline training phase hamper their real-world applicability. Approaches using online learning have shown good throughput performance, but the high memory requirement makes them inefficient or even impractical. In this paper, we propose a new effective online learning algorithm for link adaptation. Our computations show that the algorithm performs comparably to the existing online learning approaches, but ours requires minimal storage and time, which makes it more practical. Moreover it adapts to the change of channel distribution quickly.**

I. Introduction

Although proper link adaptation can improve the throughput performance of wireless systems significantly, the sheer number of environmental parameters such as signal energy, noise variance, channel state information and other subtle factors such as quantization error, non-Gaussian noise effect, and nonlinearity of systems, make it difficult to tune the transmission parameters optimally (even near optimally). Recently, there have been new flexible approaches to use machine learning algorithms for effective link adaptation. Machine learning algorithms are inherently data-driven, and rather than rely on model assumptions and approximations, they capture more environmental effects implicitly.

A. Prior Work

Online stochastic learning is used in [1] to come up with the best transmission rate for a given static channel. Although the algorithm shows good throughput performance, more considerations are needed for this algorithm to work in a rapidly changing channel. Rate adaptation in OFDM systems is considered in [2] and the prediction output is the set of SNR thresholds to switch modulation schemes. They have shown that using stochastic learning automata, the rate adaptation scheme converges to the optimal one among the predefined finite set of candidates. However, the convergence rate is closely coupled with the number of candidates, which has been rapidly growing in today's technologies. The problem is more general in [3] in the sense that the switching SNR thresholds are now arbitrary and not restricted to predefined discrete values. For a given modulation policy the performance is measured and a variant of a steepest descent algorithm is used to evolve the policy towards the optimum.

In all the literature mentioned above, either the channel is assumed to be fixed or SNR is the only link quality metric for rate adaptation. However, in a complex configuration such as MIMO-OFDM systems, a one-dimensional link quality metric is not sufficient to capture the aspects of the channel that ultimately govern the optimal choice for adaptation. Although the set of post process SNRs for all subcarriers is known to be a good metric, the required sample complexity for using such a high dimensional feature set is prohibitive, both in terms of storage and computational power required. This dimensionality problem is solved in [4] by post process SNR ordering and subselecting a particular small subset. In the prediction phase, the k nearest neighbor (kNN) algorithm is used to determine the Packet Error Rate (PER) of new channel realization and the MCS with the highest rate is chosen. In our previous work, we have improved this algorithm further by using multiclass support vector machines (SVM) with asymmetric weights. This allows us to improve performance, decrease memory requirements, and lower MCS prediction overhead.

B. Online Learning

The main two drawbacks of the two algorithms mentioned above are as follows. First, PER has to be known for each training sample *a priori* which is often impossible since in order to measure PER we need a fixed channel during several transmission tries. Second, the offline training phase requires either large memory or long training time.

Online algorithms are useful in link adaptation since the receiver sees the packets sequentially. Algorithms that do not need to relearn from scratch when the channel model changes

[1]This work was partially supported by NSF grants EFRI-0735905, CNS-0721532, CNS-0831580, and DTRA grant HDTRA1-08-0029.

slightly, or when the data arrive to a slightly modified arrival distribution, have a clear advantage in this setting. The authors in [5] have provided an online learning framework for link adaptation. They have used a modified k nearest neighbor (kNN) algorithm to learn the mappings from channel condition to PER for all the MCSs supported by the system. Every time a new packet is delivered, each MCS reports the predicted PER using kNN and the best MCS is chosen. After the transmission of the packet, the packet itself is also stored as observed data for the selected MCS for future prediction. As more packets are delivered, each MCS stores more data and due to classical consistency results for kNN (e.g., [6]), the PER prediction becomes more accurate. While statistically correct, there are algorithmic issues with this algorithm that are very important in any practical implementation. The biggest challenges of this algorithm are the resulting memory requirement and computational complexity. The kNN algorithm requires us to store all the previous samples we've seen; this is too expensive (if not prohibitively so) for a small wireless device. Moreover, the time complexity for MCS prediction also grows, and this is not favorable behavior for a real time operation that requires computational cost per operation to be bounded by a constant independent of time.

The purpose of this paper is to introduce a new learning approach, that addresses these two problems. We propose a new effective online link adaptation algorithm using online kernelized Support Vector Regression (SVR) that requires minimal size of memory and bounded time complexity for computation, yet presents a comparable performance to the existing algorithms. Moreover, our computations show that the algorithm adapts to the change of channel distribution more quickly, and hence may be more appropriate in more dynamic environments.

The remainder of this paper is organized as follows. In Section II we describe the system model and our online link adaptation algorithms. We explain how our algorithm works and what are the advantages of it. Section III provides computation results to show the performance of our algorithm. Finally, Section IV concludes this paper.

II. Online Link Adaptation in MIMO-OFDM

A. System Model

1) MIMO-OFDM System: In MIMO-OFDM systems with N_t transmit antennas and N_r receive antennas, data are transmitted over $N_s \leq \min\{N_t, N_r\}$ spatial streams. In frequency domain, a baseband signal is multiplied by a precoding matrix, then transmitted over a wireless channel. At the receiver, we use a linear equalizer to recover the transmitted signal and complex Gaussian noise is added. We assume that modulation orders and coding rates are the same for all the spatial streams, and the wireless channel is constant for all OFDM symbols in a single packet. Also, a zero forcing equalizer is assumed.

2) ARQ System: Auto Repeat Request (ARQ) is used in wireless networks to check errors in transmitted packets over wireless links. If there is no error, the receiver sends an acknowledgement message (ACK) indicating that it has correctly received a packet. Error is determined at the transmitter either when an explicit negative ACK message (NACK) is sent by the receiver or the timeout period has elapsed.

B. Learning Model

1) Frame Work: For every packet transmission, the receiver can collect the following information: Channel measurement, modulation and coding scheme and successful transmission. Since the channel measurement may not be available at the transmitter before packet transmission, the receiver determines the physical layer parameters for the next transmission and feeds back this information to the transmitter. Using the information the receiver has collected, it updates the mapping from (Channel measurement $\times$ MCS) to PER. We can approximate PER by the number of successful transmissions divided by the total number of transmissions.

Let X denote the set of all values that the channel measurement can take, and let $Y = \{0, 1\}$, where 1 represents a successful transmission. For a given MCS, we would like to use the past sequence of observations to obtain a prediction function of its future performance, i.e., its PER for some new channel measurement. Such a predictive function obtained from past data (i.e., from a sequence of observations $\mathcal{T}_t = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_t, y_t)\}$ that have used the MCS) is called a regression function. We obtain a regression function, f, that minimizes the following objective:

$$\int_{X\times Y} (f(\mathbf{x}) - y)^2 d\rho, \tag{1}$$

where ρ is a probability measure on $X \times Y$. Note that this regression function approximates the mapping from the channel measurement to success rate of transmission for the given MCS. Then given regression functions for MCSs, the receiver predicts which MCS will result in the highest throughput (success rate$\times$rate) with the current channel condition and feed back the resulting information to the transmitter for the next transmission via an ACK or a dedicated feedback message.

2) Update Rule: In this paper we use SVR, which finds the linear regression functions that minimize the mean loss function (in our case, squared error). Also, kernel machines are used to produce non-linear regression functions.

Let $k : X \times X \to \mathbb{R}$ be a Mercer kernel and $\mathscr{H}_k$ be the Reproducing Kernel Hilbert Space (RKHS) associated with k (see [7] for details). We use the following online SVR algorithm modified from [8]:

$$f_t = f_{t-1} - \gamma_t (f_{t-1}(\mathbf{x}_t) - y_t) K_{\mathbf{x}_t}\ , f_0 = 0, \tag{2}$$

where $f_t \in \mathscr{H}_k$, $K_{\mathbf{x}_t} = k(\mathbf{x_t}, \cdot)$, $\gamma_t > 0$ is a step size and $t \in \mathbb{N}$ is time. Intuitively, this algorithm can be interpreted as driving the regression function toward the actual value (y_t) from the predicted value ($f_{t-1}(\mathbf{x}_t)$). The Mercer kernel has an important property that it can be expressed equivalently as an inner product in a high dimensional RKHS, i.e., $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$, where $\phi(\cdot)$ is a mapping from the

sample space (X) to the feature space ($\mathscr{H}_k$). Then it can be easily shown that the resulting regression function f of (2) has the form of $f_t = \langle \Phi_t \boldsymbol{\alpha}_t, \cdot \rangle$, where $\Phi_t = (\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_t))$, $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_t)^\top \subseteq \mathbb{R}^t$. Therefore, updating f_t is essentially updating Φ_t and $\boldsymbol{\alpha}_t$ and the update rules are as follows.

$$\begin{aligned} f_t &= \Phi_{t-1}\boldsymbol{\alpha}_{t-1} - \gamma_t(\boldsymbol{\alpha}_{t-1}^\top \Phi_{t-1}^\top \phi(\mathbf{x}_t) - y_t)\phi(\mathbf{x}_t) \\ &= \Phi_{t-1}\boldsymbol{\alpha}_{t-1} - \gamma_t(\boldsymbol{\alpha}_{t-1}^\top \mathbf{k}_{t-1}(\mathbf{x}_t) - y_t)\phi(\mathbf{x}_t) \end{aligned}, \quad (3)$$

$$\Phi_t = (\Phi_{t-1}, \phi(\mathbf{x}_t)), \quad (4)$$

$$\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ -\gamma_t(\boldsymbol{\alpha}_{t-1}^\top \mathbf{k}_{t-1}(\mathbf{x}_t) - y_t) \end{pmatrix}, \quad (5)$$

where $[\mathbf{k}_t(\mathbf{x})]_i = k(\mathbf{x}_i, \mathbf{x})$.

3) Sparsification Algorithm: The biggest obstacle to using the update rules mentioned in the above section is that the size of Φ_t and $\boldsymbol{\alpha}_t$ grow linearly in the number of packets, thus requiring prohibitively large memory with continuous transmissions. To overcome this, we decrease the number of samples used represent f_t, by employing a sparsification algorithm proposed in [9]. The idea of this algorithm is that we only keep the linearly independent samples in $\mathscr{H}_k$ in our dictionary, i.e., the set of samples to be stored, and discard the ones that can be approximately represented as linear sums of the samples in the dictionary. At time t, we are given $t-1$ past observations, $\mathcal{T}_{t-1} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_{t-1}, y_{t-1})\}$. Suppose we have collected a subset of previous samples, $\mathcal{D}_{t-1} = \{\tilde{\mathbf{x}}_j\}_{j=1}^{m_{t-1}}$ where by construction $\{\phi(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_{t-1}}$ are linearly independent feature vectors so that all the samples up to time $t-1$ can be approximated as linear combinations of the vectors in $\mathcal{D}_{t-1}$. Now presented with a new sample $\mathbf{x}_t$, we test if $\phi(\mathbf{x}_t)$ can be represented as a linear sum of feature vectors in $\mathcal{D}_{t-1}$. To see that we first check the following condition where ν is an accuracy parameter determining the level of sparsity.

$$\delta_t = \min_{\mathbf{a}} \| \sum_{j=1}^{m_{t-1}} a_j \phi(\tilde{\mathbf{x}}_j) - \phi(\mathbf{x}_t) \|^2 \leq \nu \quad (6)$$

If condition (6) holds, $\phi(\mathbf{x}_t)$ can be approximated within a squared error ν. Condition (6) can be rewritten as the following form.

$$\delta_t = \min_{\mathbf{a}} \{ \mathbf{a}^\top \tilde{\mathbf{K}}_{t-1} \mathbf{a} - 2\mathbf{a}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + k_{tt} \}, \quad (7)$$

where $[\tilde{\mathbf{K}}_t]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, $[\tilde{\mathbf{k}}_t(\mathbf{x})]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x})$, and $k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t)$. The solution to the above minimization problem is as follows.

$$\tilde{\mathbf{a}}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \ \delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\mathbf{a}}_t \leq \nu \quad (8)$$

If the sparsification condition holds, $\phi(\mathbf{x}_t)$ can be approximated as $\phi(\mathbf{x}_t) \simeq \sum_{j=1}^{m_t} a_{t,j} \phi(\tilde{\mathbf{x}}_j)$ and the regression function f_t can be approximated as $f_t \simeq \tilde{\Phi}_t \tilde{\boldsymbol{\alpha}}_t$, where $\tilde{\Phi}_t = (\phi(\tilde{\mathbf{x}}_1), \cdots, \phi(\tilde{\mathbf{x}}_{m_t}))$, and $\tilde{\boldsymbol{\alpha}} = (\tilde{\alpha}_1, \cdots, \tilde{\alpha}_{m_t})^\top$. Since we do not expand the dictionary, $\tilde{\Phi}_t$ remains the same and the update rule for $\boldsymbol{\alpha}_t$, (5) becomes

$$\tilde{\boldsymbol{\alpha}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} - \gamma_t(\tilde{\boldsymbol{\alpha}}_{t-1}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) - y_t)\tilde{\mathbf{a}}_t. \quad (9)$$

If the sparsification condition does not hold, we update the dictionary as $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$ and the update rules, (4,5) may be rewritten as follows.

$$\tilde{\Phi}_t = (\tilde{\Phi}_{t-1}, \phi(\mathbf{x}_t)) \quad (10)$$

$$\tilde{\boldsymbol{\alpha}}_t = \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ -\gamma_t(\tilde{\boldsymbol{\alpha}}_{t-1}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) - y_t) \end{pmatrix} \quad (11)$$

Also, by the following recursive procedure, we do not have to compute $\tilde{\mathbf{K}}_t^{-1}$ every time we expand our dictionary, $\mathcal{D}_t$.

$$\begin{aligned} \tilde{\mathbf{K}}_t &= \begin{pmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k_{tt} \end{pmatrix} \Rightarrow \\ \tilde{\mathbf{K}}_t^{-1} &= \frac{1}{\delta_t} \begin{pmatrix} \delta_t \tilde{\mathbf{K}}_{t-1}^{-1} + \tilde{\mathbf{a}}_t \tilde{\mathbf{a}}_t^\top & -\tilde{\mathbf{a}}_t \\ \tilde{\mathbf{a}}_t^\top & 1 \end{pmatrix} \end{aligned} \quad (12)$$

III. Computations

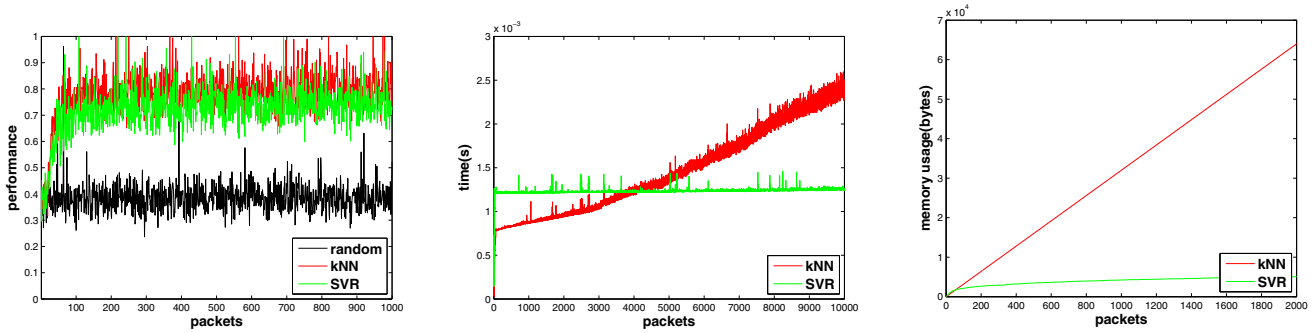
In this section, we evaluate the performance of our algorithm using IEEE 802.11n based simulation study. We use the packet error rate simulation data of [10]. Under 2×2 MIMO-OFDM and 4 taps frequency selective fading, 56,000 channels are generated according to the zero-mean complex-Gaussian distribution with SNR varying from 0 to 27 and PER is simulated for every pair of channel realization and MCS. We use this PER data to generate random transmissions with success probability of (1-PER). Following the feature set extraction scheme shown in [4], we use four dimensional feature space of ordered post process SNR. For all the performance figures throughout this paper, the x axis represents the sequential index of arriving packets and the y axis represents the relative throuput performance of a given algorithm versus the performance of an ideal algorithm that chooses MCS with the best throuput (i.e. (1-PER)×rate).

A. Online SVR vs. online kNN

As shown in Fig. 1a the performance of our algorithm is comparable to that of online kNN, but we can see in Fig. 1b and Fig. 1c that our algorithm generally requires smaller memory and time overhead. More importantly, we can see that memory usage and time consumption remain constant with our algorithm whereas they increase monotonically with online kNN which makes our algorithm more favorable to real time applications such as link adaptation.

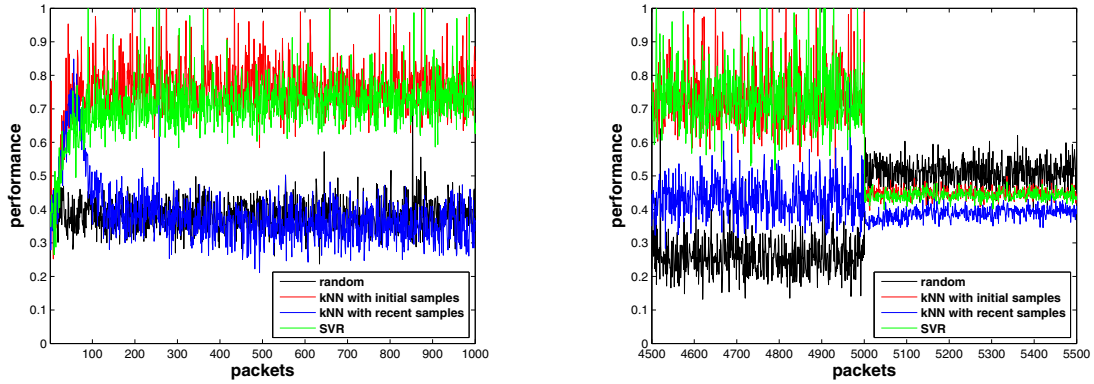
B. Online SVR vs. online local kNN

What if we limit the memory usage of online kNN to match it with that of our algorithm? That is, instead of storing all the previous samples, we keep only a small part of the sample set. We compare our algorithm with online kNN with initial samples (i.e., we store first certain number of samples and discard all the samples after then), and online kNN with most recent samples (i.e., everytime we see a new sample, we drop the oldest sample and keep the newest one.) Interestingly, we can see in Fig. 2a that online kNN with initial samples works well throughout the simulation but online kNN with most recent samples works very poorly after some point. This is because we choose an MCS with best empirical performance



(a) throughput performance (b) time complexity (c) memory requirement

Fig. 1. online SVR vs. online kNN



(a) throughput performance over static channel distribution

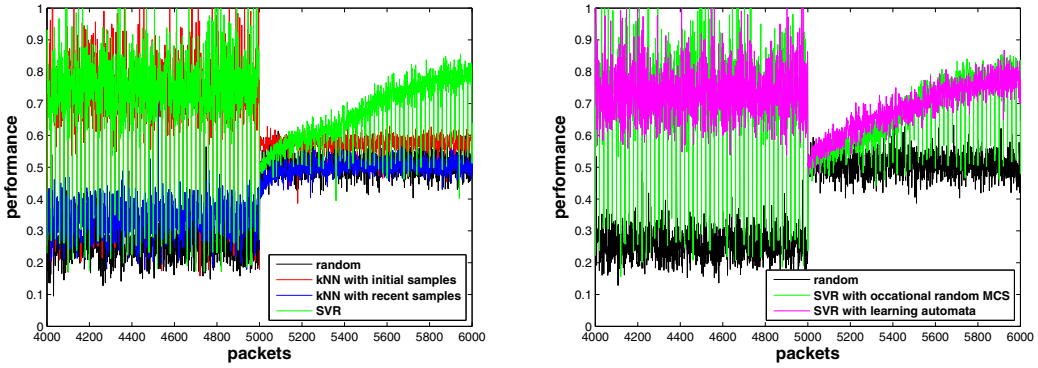
(b) throughput performance over sudden change of channel distribution

Fig. 2. online SVR vs. online kNN with initial samples vs. online kNN with recent samples

every time we're given a new sample. At initial stage, since we don't have enough data, the prediction is not accurate, hence some samples are assigned to the wrong MCSs and the set of initial samples for each MCS has variety of channel realizations by errornous predictions and MCS assignments. However, after some point, samples for each MCS are biased towards the ones that are appropriate for the MCS, thus lose the diversity and the prediction for the channel realizations that are far from those samples becomes very poor. This means if we want to limit the number of data for online kNN, we need to keep the diversity of sample positions for every MCS, which can be achieved by choosing initial samples. However, when the distribution of channel changes, keeping initial samples for online kNN may not be the best solution since it can't capture that change of distribution. To compare how different algorithms work in the presence of the change of channel distribution, we conduct a simulation, in which we have channel realizations with SNR from 0 to 13 at the first half and ones with SNR from 14 to 27 at the second half. Fig. 2b shows that all the algorithms including ours work poorly after the distribution change. This is because we use an aggressive exploitation policy (i.e., we choose the best MCS and never look into the others). Until the time that the distribution changes, only the MCSs with slower rates are properly updated since we have had channels with lower SNRs and even when we're getting higher SNR channels, the MCSs with higher rates never get chances to update their regression functions since they falsely report that success rates are low with them hence never be used. Therefore we keep using the slower ones. In the next section, we try different exploration/exploitation policies and compare the algorithms with them.

C. Exploration vs. exploitation

We first apply a simple exploration policy that every several samples, we choose random MCS regardless of channel realization. We compare our algorithm with online kNN with initial samples and online kNN with most recent samples. As shown in Fig. 3a, only our algorithm quickly approaches to the best throughput performance. The reasons are as follows. First, since online kNN with initial samples doesn't update data, no exploration can help after the limited size of buffers are already filled with initial samples and the algorithm can't adapt to the change of channel distribution. Second, online kNN with most recent samples work poorly by the same reason in the case of static channel distribution in the previous section. Lastly, since SVR accumulates the effects of all the samples up to the



(a) throughput performance with occasional random MCS selection

(b) throughput performance with learning automata

Fig. 3. different exploration/exploitation policies over sudden change of channel distribution

current time (without actually storing all the data), it takes the best of both: Random MCS explorations using recent samples let our algorithm to properly update regression functions for higher rate MCSs, yet the data set preserves the diversity of channel samples by the effect of initial samples.

One possible drawback of this exploration policy is periodic performance drop due to random MCS selection. Additionally we have implemented another exploration policy so called stochastic learning automata [11]. It updates the MCS selection probability vector so that the probability to choose the MCS with the best empirical performance is the highest and converges to 1 as we see more and more positive feedback that the MCS is the best one. This policy is less aggressive than the one that always chooses the MCS with the best performance and helps us to avoid losing the chance to select the best MCS completely due to some successive unlucky events at the initial stage. The computation results in Fig. 3b shows that online SVR with learning automata results in smoother performance curve without periodic performance drops. However, we haven't noticed any increase of average throughput performance, so in terms of performance this exploration policy has no advantage over simple random MCS selection.

IV. Conclusions

In this paper we have shown that online SVR with sparsification is suitable for real time link adaptation in MIMO-OFDM wireless systems due to its bounded memory usage and time complexity as opposed to other algorithms that show monotonically increasing memory usage and time complexity as the number transmissions increases. Also with proper exploration policy our algorithm quickly adapts to a new environment. These advantages allow us to use link adaptation in different situations without offline training from the scratch every time we encounter a new environment.

Issues we have not mentioned include the channel estimation error at the receiver and feedback delay. Perfect channel state knowledge at the receiver and no feedback delay are assumed in this paper. Robust optimization techniques or risk measure approaches can be used to meet users' need. Another extension is the online feature extraction. Although we're using a fixed feature set, different environment may require different feature set extraction. It would be nice to have an algorithm to extract the feature space that represents the characteristics of data, yet have low dimension in real time. Online Principal Component Analysis may work in this case.

References

[1] M. A. Haleem and R. Chandramouli, "Adaptive stochastic iterative rate selection for wireless channels," *IEEE Communications Letters*, vol. 8, pp. 292-294, May., 2004.

[2] A. Misra, V. Krishnamurthy and S. Schober, "Stochastic learning algorithms for adaptive modulation," *IEEE Workshop on Signal Processing Advances in Wireless Communications*, pp. 756-760, 2005.

[3] C. Tang and V. Stolpman, "An adaptive learning approach to adaptive OFDM," *Proc. IEEE Wireless Communications and Networking Conference*, vol. 3, pp. 1406-1410, Mar., 2004.

[4] R. C. Daniels, C. M. Caramanis and R. W. Heath Jr., "Adaptation in Convolutionally-Coded MIMO-OFDM Wireless Systems through Supervised Learning and SNR Ordering," *IEEE Tran. on Vehicular Technology*, vol. 59, pp. 114-126, Jan., 2010.

[5] R. C. Daniels and R. W. Heath Jr., "An online learning framework for link adaptation in wireless networks," *Information Theory and Applications Workshop*, pp. 138-140, Feb., 2009.

[6] C. J. Stone, "Consistent Nonparametric Regression," *Ann. Statist.*, vol. 5, pp. 595-645, 1977.

[7] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 1384-5810, Jun., 1998.

[8] S. Smale and Y. Yao, "Online Learning Algorithms," *Foundations of Computational Mathematics*, vol. 6, pp. 1615-3375, Apr., 2006.

[9] Y. Engel, S. Mannor and R. Meir, "The Kernel Recursive Least Squares Algorithm," *IEEE Tran. on Signal Processing*, vol. 52, pp. 2275-2285, Aug., 2004.

[10] R. C. Daniels, C. M. Caramanis and R. W. Heath Jr., "PER Simulations for 2x2 IEEE 802.11n Systems," http://128.83.198.111/mlearn/4_tap_files_index.htm, 2008.

[11] M. A. L. Thathachar and P. S. Sastry, "Networks of Learning Automata," *Kluwer Academic Publishers*, 2004.