Graph Filter Design for Distributed Network Processing: A Comparison between Adaptive Algorithms

Atiyeh Alinaghi, Stephan Weiss, Vladimir Stankovic, Ian Proudler
Dept. of Electronic & Electrical Engineering, University of Strathclyde, Glasgow, UK
{atiyeh.alinaghi, stephan.weiss, vladimir.stankovic, ian.proudler}@strath.ac.uk

2021 Sensor Signal Processing for Defence Conference (SSPD) | 978-1-6654-3314-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/SSPD51364.2021.9541468

***Abstract*—Graph filters (GFs) have attracted great interest since they can be directly implemented in a diffused way. Thus it is interesting to investigate GFs to implement signal processing operations in a distributed manner. However, in most GF models, the input signals are assumed to be time-invariant, *static*, or change at a very low rate. In addition to that, the GF coefficients are usually set to be *node-invariant*, i.e. the same for all the nodes. Yet, in general, the input signals may evolve with time and the underlying GF may have parameters dependent on the nodes. Therefore, in this paper, we consider *dynamic* input signals and both types of GF coefficients, *node-variant*, i.e. vary on different nodes, and node-invariant. Then, we apply LMS and RLS algorithms for GF design, along with two others called adapt-then-combine (ATC) and combined RLS (CRLS) to estimate the GF coefficients. We study and compare the performance of the algorithms and show that in the case of node-invariant GF coefficients, CRLS gives the best performance with lowest mean-square-displacement (MSD), whereas, for node-variant case, RLS represents the best results. The effect of bias in the input signal has also been examined.**

***Index Terms*—Graph Signal Processing, Graph Filtering, Distributed Processing, Adaptive algorithms**

I. Introduction

In order to infer information from sensor networks, conventional *centralized processing* is often used but may require high transmission power, large communication bandwidth, and costly energy consumption in the central unit. Therefore, a *distributed* approach has been suggested where each node only communicates with its neighbouring nodes, exchanging the information locally, and has its own light processing unit [1]–[4].

A sensor network can be modelled as a graph where the sensors are represented as graph nodes and the inter-sensor communications links are denoted as graph edges. As such, we can apply graph signal processing (GSP) methods, which has emerged recently to extend the classical signal processing concepts to the signals on the vertices of a graph [5]. Spectra of graphs, graph filters (GFs) and other tools to process graph signals are defined accordingly [6], [7]. A practical benefit of GFs is that they may be directly implemented in a distributed manner [8]–[12].

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/S000631/1 and the MOD University Defence Research Collaboration in Signal Processing.

GSP has been applied to various applications ranging from social media webs, brain activity connections and wireless sensor networks, to power grids and transportation networks [6]. For defence applications, one can imagine situations where a number of sensors, such as sonobuoys or disposable RF connected sensors, are deployed to communicate and infer information based on the observations and measurements. These sensors can be considered as the nodes in a graph and then GSP methods, such as GF, can be applied to analyse the data more efficiently in a distributed fashion.

Graph filtering is one of the core tools in GSP, which can be exploited to represent a transformation of the graph input signals.In the general case, a GF is a transformation matrix mapping the input signal, $\mathbf{x}$, to the output. Therefore, any linear transformation, $\mathbf{T}(\mathbf{x}) = \mathbf{B}\mathbf{x}$, can be implemented by a GF. Since the structure of the network, the graph topology, is fundamental in distributed processing, it is required to incorporate the graph shift operator, $\mathbf{S}$ (a matrix representing the graph structure) in the GF design. In this paper, we assume that $\mathbf{B}$ is shift invariant with respect to $\mathbf{S}$ and can be represented as a polynomial of $\mathbf{S}$, i.e. $\mathbf{S}$ is shift enabled [11].

In most models such as [12]–[14], the input signals on the nodes are assumed to be static, i.e. time-invariant, which is not always true. Therefore, in [9], a GF model is introduced which incorporates the time-varying behaviour of the signals. Even though this model is more general, the authors still assume that the GF coefficients are node-invariant, i.e. are the same on different nodes. On the other hand, in [13], node-variant filters are proposed such that the coefficients vary from node to node. It gives more flexibility which enables the design of more general filters with lower orders. Nevertheless, the model is for time-invariant inputs. Therefore, to be more generic, we consider a node-variant GF model, with time-variant inputs.

Having decided on the GF model, we need an algorithm to estimate the GF coefficients. Given the reference system we then use adaptive GF algorithms to estimate the coefficients. First, we consider an adapt-then-combine (ATC) least mean squares (LMS) algorithm which combines the estimated coefficients on the neighbouring nodes [9]. We compare this to an LMS algorithm that operates independently on each node, which is equivalent to ATC without its combine step, and therefore permits a node-variant solution. In addition to

that, we apply the recursive least square (RLS) algorithm in [14] to use it for graph filter design by integrating the graph shift operator (GSO) in the model. As in [14], we also add a combine step to RLS, called combine RLS (CRLS), which makes it comparable to ATC. Ultimately, we implement these approaches and compare their performance with biased and unbiased time-variant input signals.

In the following, we start with a review of graph filter design in Section II. Then in Section III we explain adaptive filters on graphs. Section IV is dedicated to experimental results, while conclusions are drawn in Section V.

II. Graph Filter Design and Implementation

Various approaches to designing a distributed system are possible. Here we envisage finding the coefficients off-line for the GF design. Then the GF and hence $\mathbf{T}(\mathbf{x})$ can be implemented in a distributed manner.

A. Graph Filter Models

Let us consider an undirected graph $\mathcal{G}$ with a set of N nodes or vertices, $\mathcal{N}$, and a set of edges or links, $\mathcal{E}$, such that if node i is connected to node j, then $(i,j) \in \mathcal{E}$. The neighbourhood of node i is defined as the set of nodes $\mathcal{N}_i = \{j|(i,j) \in \mathcal{E}\}$ connected to i including node i itself. For any given graph, an adjacency matrix, $\mathbf{A}$, is defined as an $N \times N$ square matrix with non-zero elements $a_{ji} \in \mathbb{R}$ if and only if $(i,j) \in \mathcal{E}$. For each graph, there exists a GSO, $\mathbf{S}$, which can be chosen as $\mathbf{A}$, the adjacency matrix, or $\mathbf{L}$, the Laplacian matrix [6], [7]. In this paper we will be using the adjacency matrix as the shift operator.

In GSP applications, one is interested in the analysis of signals on the graph, defined by $\mathbf{x} = [x_1, x_2, ..., x_N]^T \in \mathbb{R}^N$, where x_k represents the value of the signal at node k. The graph signal at discrete time t is denoted by $\mathbf{x}(t)$. In distributed processing problems, we have a transform, $\mathbf{T}$, which maps the input graph signals, $\mathbf{x}(t)$, to the desired outputs on the nodes, $\mathbf{y}(t) = \mathbf{T}(\mathbf{x}(t))$. Recall that $\mathbf{T}(\mathbf{x})$ can be represented as a polynomial in the shift matrix. Our goal is to model this transform as a GF.

In this section we start with node-variant GFs with time-varying inputs. We also assume that the shift operation takes a non-negligible amount of time with the same delay as the sampling period of the input signal. In this node-variant case, the GF coefficients are different on each node, k, and can be put in an $M \times N$ matrix $\mathbf{D}$ with each element $\mathbf{D}(m,k) = h^o_{m,k}$, being the optimal mth coefficient on the kth node. Then, an N element vector $\boldsymbol{h}^{(m)}$ can be defined as the mth row of $\mathbf{D}$, while the kth column of $\mathbf{D}$ can be symbolized by an $M \times 1$ vector $\boldsymbol{h}^o_k$. The filter model can be then described as:

$$\mathbf{y}(t) = \sum_{m=0}^{M-1} \text{diag}(\boldsymbol{h}^{(m)}) \mathbf{S}^m \mathbf{x}(t-m) + \mathbf{v}(t), \qquad (1)$$

where M is the order of the graph filter, and $\mathbf{v}(t) = [v_1(t), v_2(t), ..., v_N(t)]^T$ is multivariate Gaussian distributed noise which may not be always present, but we include that to be comparable to the model in [9].

For the simplest case where the coefficients are node-invariant, all the N elements of $\boldsymbol{h}^{(m)}$ will be the same on different nodes, and so $\text{diag}(\boldsymbol{h}^{(m)}) = \mathbf{I}_N h^o_m$, $h^o_m \in \mathbb{R}$. Thus for the time-invariant input and node-invariant filter, the equation can be written as:

$$\mathbf{y}(t) = \sum_{m=0}^{M-1} h^o_m \mathbf{S}^m \mathbf{x}(t) + \mathbf{v}(t). \qquad (2)$$

In other words, the general model (1) becomes time-invariant just by setting $\mathbf{x}(t-m) = \mathbf{x}(t)$ for all m, and node-invariant by setting $h^o_{m,k} = h^o_m$ for all k, enforcing the coefficients to be the same on different nodes as in (2).

III. Adaptive Filters on Graphs

Based on (1) for the general node-variant case, the output signal on each node, say kth node, $y_k(t)$, can be modelled separately as:

$$y_k(t) = \sum_{m=0}^{M-1} h^o_{m,k} [\mathbf{S}^m \mathbf{x}(t-m)]_k + [\mathbf{v}(t)]_k, \qquad (3)$$

where $[.]_k$ represents the kth row of a vector.

Similar to [9], we define vector $\mathbf{z}(t-m) \triangleq \mathbf{S}^m \mathbf{x}(t-m)$, and then an $M \times 1$ vector $\mathbf{z}_k(t)$ as:

$$\mathbf{z}_k(t) \triangleq col\{[\mathbf{z}(t)]_k, [\mathbf{z}(t-1)]_k, ..., [\mathbf{z}(t-M+1)]_k\}. \qquad (4)$$

Then (3) can be written alternatively as:

$$y_k(t) = \mathbf{z}^T_k(t) \boldsymbol{h}^o_k + v_k(t), \qquad t \geq M-1. \qquad (5)$$

where the optimum $\boldsymbol{h}^o_k$ is the kth column of $\mathbf{D}$, and the adaptive filter model is $\mathbf{z}^T_k(t) \boldsymbol{h}_k$. We then define a global cost function:

$$\mathbf{J}(\mathbf{D}) = \sum_{k=1}^{N} \mathbf{J}_k(\boldsymbol{h}_k), \qquad (6)$$

where $\mathbf{J}_k(\boldsymbol{h}_k)$ is the local cost function at node k which can be mean square error for LMS based algorithms:

$$\mathbf{J}_k(\boldsymbol{h}_k) = \mathbb{E}\{|y_k(t) - \mathbf{z}^T_k(t) \boldsymbol{h}_k|^2\}, \qquad (7)$$

or weighted least squares error function for RLS based algorithms:

$$\mathbf{J}_k(\boldsymbol{h}_k) = \sum_{i=1}^{t} \lambda^{t-i} |y_k(i) - \mathbf{z}^T_k(i) \boldsymbol{h}_k|^2, \qquad (8)$$

where λ is a forgetting factor. This optimization problems are to be solved using adaptive methods as described in the following sections.

A. LMS-Based Algorithms

In [9] the ATC approach is proposed to estimate the filter coefficients iteratively:

$$\begin{cases} \boldsymbol{\psi}_k(t+1) = \boldsymbol{h}_k(t) + \mu_k \mathbf{z}_k(t)(y_k(t) - \mathbf{z}_k^T(t)\boldsymbol{h}_k(t)), \\ \boldsymbol{h}_k(t+1) = \sum_{l\in\mathcal{N}_k} c_{lk}\boldsymbol{\psi}_l(t+1), \end{cases} \tag{9}$$

where $\boldsymbol{h}_k(t)$ is the vector of filter coefficients updated at time t, μ_k the step-size, and c_{lk} are non-negative combination factors:

$$c_{lk} > 0, \quad \sum_{l=1}^{N} c_{lk} = 1 \quad \text{and} \quad c_{lk} = 0 \quad \text{if} \quad l \notin \mathcal{N}_k. \tag{10}$$

In other words, $\mathbf{C} \triangleq [c_{lk}]$ is a left-stochastic matrix and $\boldsymbol{\psi}_k(t+1)$ is an $M \times 1$ intermediate vector to represent the two step algorithm. As a result, the coefficients will be averaged over the nodes, converging to the same values on different nodes. Thus, the ATC is intended for a problem with node-invariant coefficients.

To modify this algorithm to be node-variant we effectively remove the combine step by setting the combination matrix $\mathbf{C}$ to an identity matrix $\mathbf{I}$.

B. RLS-Based Algorithms

In this section, we apply the RLS algorithm for use with a graph. We incorporate the structure of the underlying graph of the network by including the GSO, $\mathbf{S}$. Moreover, we estimate distinctive parameters (GF coefficients) on each node. Even though the RLS algorithm has previously been applied to a sensor network problem in [14], the authors did not include the shift operator in the model.

To perform an RLS algorithm, the weighted sample covariance matrix at each time is required:

$$\hat{\mathbf{R}}_{\mathbf{z}_k}(t) = \sum_{i=1}^{t} \lambda^{t-i}\mathbf{z}_k(i)\mathbf{z}_k^T(i) + \lambda^t \delta \mathbf{I}_M, \tag{11}$$

where $\mathbf{z}_k(t)$ is the vector defined in (4) which can also be represented as:

$$\mathbf{z}_k(t) \triangleq [\mathbf{x}(t), \mathbf{S}\mathbf{x}(t-1), ..., \mathbf{S}^{M-1}\mathbf{x}(t-M+1)]_k^T. \tag{12}$$

The parameter, λ, is the forgetting factor and δ is a small positive number that serves as a regularization parameter. We need the inverse matrix $\mathbf{P}_k(t) = \hat{\mathbf{R}}_{\mathbf{z}_k}^{-1}(t)$ which is not feasible to calculate in a compact form. Therefore, we update it iteratively along with the coefficients. The following algorithm is RLS with a combine step which we call CRLS. This is a modified version of Diffusion BC-RLS algorithm in [14], but with the GSO, $\mathbf{S}$, incorporated in the regressor, $\mathbf{z}_k(t)$, and no compensation step:

$$\begin{aligned} \mathbf{P}_k(t+1) &= \lambda^{-1}\left(\mathbf{P}_k(t) - \frac{\lambda^{-1}\mathbf{P}_k(t)\mathbf{z}_k(t)\mathbf{z}_k^T(t)\mathbf{P}_k(t)}{1+\lambda^{-1}\mathbf{z}_k^T(t)\mathbf{P}_k(t)\mathbf{z}_k(t)}\right) \\ \boldsymbol{\phi}_k(t+1) &= \boldsymbol{h}_k(t) + \mathbf{P}_k(t)\mathbf{z}_k(t)(y_k(t) - \mathbf{z}_k^T(t)\boldsymbol{h}_k(t)), \\ \boldsymbol{h}_k(t+1) &= \sum_{l\in\mathcal{N}_k} c_{kl}\boldsymbol{\phi}_k(t+1). \end{aligned} \tag{13}$$

Similar to ATC, CRLS has a combine step with combination matrix $\mathbf{C}$, which averages the coefficients across the neighbouring nodes. Therefore, it is also a node-invariant algorithm. For the node-variant case we effectively remove the combine step by setting the $\mathbf{C}$ to an identity matrix $\mathbf{I}$. This results in a node-variant RLS algorithm.

C. Effect of Biased Graph Signals

The data that has been addressed in various other publications [9], [13], [14], are often assumed to be zero-mean. It is possible that the graph signal, $\mathbf{x}(t)$, is not zero mean [9]. Therefore, we briefly investigate the effect that a bias term in the data has on the above algorithms. In this case there is a bias term, $\mathbf{b} \in \mathbb{R}^N$ added to the data, such that $\mathbf{x}_b(t) = \mathbf{x}(t) + \mathbf{b}$. We have $\mathbb{E}\{\mathbf{x}_b\} = \mathbf{b}$, so that, $\mathbf{R}_b = \mathbf{R} + \mathbf{b}\mathbf{b}^H$, where $\mathbf{b}\mathbf{b}^H$ is a rank-one matrix. As a result, the bias term changes the eigenvalues of the covariance matrix, and generally increases both the signal power and the condition number. Since every value in $\mathbf{R}_b$ is larger than the equivalent values in $\mathbf{R}$, we find that the covariance matrix of non-zero-mean data will contain a larger estimation error [15]. As the bias increases the condition number, it will deform the mean-squared-error (MSE) cost function, and gradient-based methods such as the LMS may converge slower. For RLS-type algorithms, this difference in convergence will be less pronounced due to the gradient correction via the estimate of the inverse covariance matrix.

IV. Experimental Results

Similar to [9], we generated random connected Erdös-Renyi graphs, with $N = 20$ nodes; and example of which is shown in Fig. 1. Using a similar construction as in [9], "this graph was obtained by generating an $N \times N$ symmetric shift matrix, $\mathbf{S}$, whose entries were governed by Gaussian distribution $\mathcal{N}(0,1)$ and then threshold edges to be between 1.2 and 1.8 in absolute value to yield an effective probability of an edge $p \approx 0.07$. The edges were soft thresholded by 1.1 to be between 0.1 and 0.7 in magnitude." The shift matrix, $\mathbf{S}$, was then normalized by 1.1 times its largest eigenvalue to prevent instability. It ensures that $\mathbf{S}^m$ contracts for increasing m, and hence does not cause stability issues.

In [9], the authors assumed that the graph signal, $\mathbf{x}(t)$, is i.i.d Gaussian with zero-mean and covariance matrix $\mathbf{R}_\mathbf{x}$, where $\mathbf{R}_\mathbf{x}$ was chosen as the solution of the Lyapunov equation $\mathbf{S}\mathbf{R}_\mathbf{x}\mathbf{S}^T - \mathbf{R}_\mathbf{x} + \mathbf{I} = 0$. This is equivalent to generating the signals through a Gauss-Markov model with the shift operator, $\mathbf{S}$, as its transfer function. The logic behind this choice is that usually in real situations the input signals on the neighbouring nodes are somehow related.

To investigate the effect of bias, we added a DC component to the input signal, which was chosen to be 2.5 times the input variance $\sigma_\mathbf{x}^2$. The factor 2.5 is selected based on the convergence behaviour of the algorithms. In order to be comparable to [9],"the noise $\mathbf{v}(t)$ was also set to zero-mean Gaussian with covariance $\mathbf{R}_\mathbf{v} = \text{diag}\{\sigma_{v,k}^2\}_{k=1}^N$, where the variances $\sigma_{v,k}^2$ were randomly generated from uniform

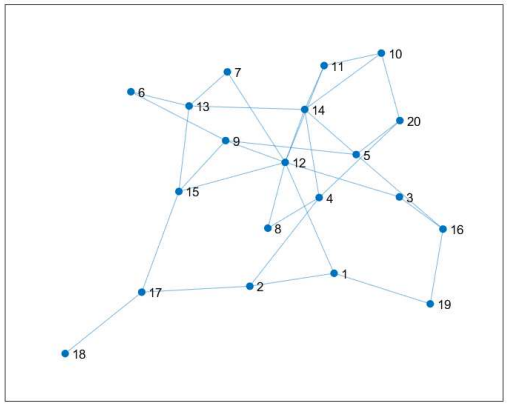

Fig. 1. A random Erdos-Renyi graph with $N = 20$ nodes.

distribution $\mathcal{U}(0.1, 0.15)$. The filter order M was set to 3 and the coefficients $h^o_{m,k}$ of the target filter, i.e $\mathbf{T}(\mathbf{x})$ were randomly drawn from a uniform distribution $\mathcal{U}(0, 1)$."

For LMS and ATC algorithms, the step-size, μ, was chosen as $1/\zeta_{max}$, where ζ_{max} is the maximum eigenvalue of the covariance matrix $\mathbf{R}_{\mathbf{z}_k}$ as in (11) with $i = t$, $\lambda = 1$ and $\delta = 0$. For RLS and CRLS algorithms, the forgetting factor, λ, was selected as 0.9999. The combine factors for ATC [9], and CRLS were set to $c_{lk} = 1/|\mathcal{N}_k|$ for $l \in \mathcal{N}_k$, where $|.|$ denotes the cardinality of its entry.

We ran 200 Monte-Carlo simulations and for each simulation a new random Erdös-Renyi graph was generated. Since the explained procedure does not guarantee the connectivity of the generated random graphs, we discarded all the graphs that were not connected. Along with that, at each simulation, a new input signal was generated based on the new shift matrix. The ground truth filter coefficients were also randomly generated at each run. After each simulation the mean-squared-displacement (MSD)s, $\mathbb{E}\{||\boldsymbol{h}^o_k - \boldsymbol{h}_k||^2\}$, between the ground truth coefficients and the estimated parameters were calculated for each node, k, and then after 200 runs, averaged over all the simulations. This MSD metric calculates the deviation of each estimated parameter from the ground truth coefficients and so demonstrates how well a system has been identified.

We set the ground truth coefficients, $\boldsymbol{h}^o_k$, to be either node-invariant or node-variant. Then we ran the algorithms for both biased and unbiased input signals. Fig. 2 shows an example MSD of the estimated coefficients on all the nodes, $k = 1, \ldots, N$, for one of the coefficients, m, where the ground truth was node-invariant and input was biased. We can see that CRLS and ATC show the best performance with the lowest MSD, while RLS has moderate performance, and LMS results are not very appealing. It is not surprising since ATC and CRLS combine the coefficients at each iteration steps which results in an average value on different nodes, or in other words, it results in node-invariant estimations. The algorithms typically suffer from gradient noise, in particular the LMS type ones. Additional averaging over the graph dimension helps to reduce its effect.

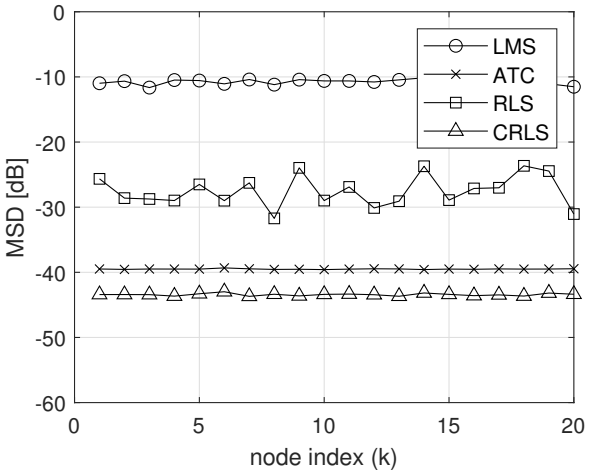

Fig. 2. MSD between the node-invariant ground truth GF coefficients and the estimated values applying, LMS, ATC, RLS, and CRLS algorithms with biased input on all the nodes $k = 1, \ldots, N$, averaged over 200 Monte-Carlo simulations in dB.

Table I represents the averaged MSD over all the nodes for the four different scenarios, node-invariant and node-variant ground truths with unbiased and biased inputs. We can see that for node-invariant ground truth, ATC and CRLS show better performance with lower MSD, while LMS shows the worst results. This is reasonable as in the ATC and CRLS there is a combine step which gets the average of the coefficients to be the same on all the nodes. On the contrary, for node-variant case, ATC and CRLS display higher, worse, MSDs, while RLS gives the lowest MSD. It is also noticeable that the performance of LMS for both node-variant and invariant cases are more or less the same. Similar behaviour can also be observed for RLS algorithm. Therefore, for real data and applications, where the nature of underlying GF model is not necessarily known, RLS, can be a better choice since it shows robustness under four different conditions.

By looking at Table I, we can also see that the bias has considerable effect on LMS, increasing its MSD. However, the main impact of the bias is the convergence speed of the error shown in Fig. 3, which is calculated on each node as:

$$e_k(t) = \mathbb{E}\{|y_k(t) - \sum_{m=0}^{M-1} h_{m,k}[\mathbf{S}^m \mathbf{x}(t-m)]_k|^2\}, \qquad (14)$$

where $\mathbb{E}\{\}$ here represents an ensemble average over 200 Monte-Carlo simulations. As seen in Fig. 3, the LMS and ATC algorithms, converge considerably faster when the input signal is unbiased. Bias also increases the convergence time for RLS and CRLS, but it is not that pronounced.

V. Conclusion

In this paper a general graph filter (GF) model has been considered, which can be applied for both time-invariant and

TABLE I

MSD VALUES BETWEEN THE GROUND TRUTH AND THE ESTIMATIONS AVERAGED OVER ALL THE NODES, k, AND COEFFICIENTS, m, AFTER 200 MONTE-CARLO SIMULATIONS IN [DB].

	node-invariant		node-variant	
	unbiased	biased	unbiased	biased
LMS	-22.72	-15.81	-22.26	-15.76
ATC	-39.39	-42.33	(-10.74)	(-8.78)
RLS	-31.38	-32.60	-30.82	$\mathbf{-32.79}$
CRLS	-46.37	$\mathbf{-48.03}$	(-10.91)	(-10.89)

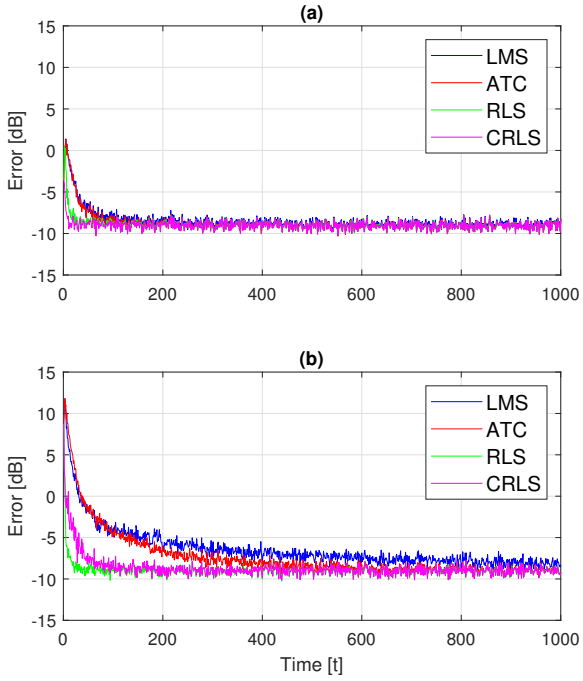

Fig. 3. Error between the desired output and the estimated output after each iteration on a random node, with node-invariant coefficients for (a) unbiased and (b) biased inputs.

time-varying inputs. In other words, it integrates the dynamic behaviour of the input signal. In addition to that, the model is modified to be compatible for both node-invariant, where the parameters are the same on different nodes, and node-variant, where the parameters are node dependent. Then, four different adaptive algorithms, namely LMS, ATC, RLS and, CRLS, have been applied to estimate the model parameters, GF coefficients, by minimizing a cost function. By comparing the algorithms, CRLS has shown the best performance for node-invariant case, while the RLS has given the lowest MSD for node-variant coefficients. The impact of bias in the input signal has also been examined, showing the most deterioration for LMS algorithms. The bias has also shown a considerable effect on the convergence of LMS and ATC algorithms by decreasing their convergence speed. Having considered all the factors, RLS seems to be a better choice as it shows robust performance under different conditions.

REFERENCES

[1] C. G. Lopes and A. H. Sayed, "Diffusion least-mean squares over adaptive networks: Formulation and performance analysis," *IEEE Transactions on Signal Processing*, vol. 56, no. 7, pp. 3122–3136, 2008.

[2] A. Bertrand and M. Moonen, "Distributed adaptive node-specific signal estimation in fully connected sensor networks—part i: Sequential node updating," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5277–5291, 2010.

[3] ——, "Distributed adaptive node-specific signal estimation in fully connected sensor networks—part ii: Simultaneous and asynchronous node updating," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5292–5306, 2010.

[4] F. de la Hucha Arce, M. Moonen, M. Verhelst, and A. Bertrand, "Distributed adaptive node-specific signal estimation in a wireless sensor network with noisy links," *Signal Processing*, vol. 166, p. 107220, 2020.

[5] E. Isufi, G. Leus, and P. Banelli, "2-dimensional finite impulse response graph-temporal filters," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2016, pp. 405–409.

[6] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[7] L. Stankovic, D. P. Mandic, M. Dakovic, I. Kisil, E. Sejdic, and A. G. Constantinides, "Understanding the basis of graph signal processing via an intuitive example-driven approach [lecture notes]," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 133–145, 2019.

[8] R. Nassif, C. Richard, J. Chen, and A. H. Sayed, "A graph diffusion lms strategy for adaptive graph signal processing," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 1973–1976.

[9] ——, "Distributed diffusion adaptation over graph signals," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4129–4133.

[10] D. G. Tiglea, R. Candido, and M. T. M. Silva, "An adaptive sampling technique for graph diffusion lms algorithm," in *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, pp. 1–5.

[11] L. Chen, S. Cheng, V. Stankovic, and L. Stankovic, "Shift-enabled graphs: Graphs where shift-invariant filters are representable as polynomials of shift operations," *IEEE Signal Processing Letters*, vol. 25, no. 9, pp. 1305–1309, 2018.

[12] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2320–2333, 2019.

[13] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Transactions on Signal Processing*, vol. 65, no. 15, pp. 4117–4131, 2017.

[14] A. Bertrand and M. Moonen, "Distributed adaptive estimation of node-specific signals in wireless sensor networks with a tree topology," *IEEE Transactions on Signal Processing*, vol. 59, no. 5, pp. 2196–2210, 2011.

[15] C. Delaosa, J. Pestana, N. J. Goddard, S. Somasundaram, and S. Weiss, "Sample space-time covariance matrix estimation," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8033–8037.

[16] Yinman Lee and Wen-Rong Wu, "An lms-based adaptive generalized sidelobe canceller with decision feedback," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 3, 2005, pp. 2047–2051 Vol. 3.

[17] F. Hua, R. Nassif, C. Richard, H. Wang, and A. H. Sayed, "A preconditioned graph diffusion lms for adaptive graph signal processing," in *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 111–115.

[18] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Filtering random graph processes over random time-varying graphs," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4406–4421, 2017.

[19] S. Haykin, *Adaptive Filters*. NJ, Englewood Cliffs: Prentice-Hall, 1995.