Proceedings of Machine Learning Research vol 144:1–14, 2021

Graph Neural Networks for Distributed Linear-Quadratic Control

Fernando Gama* FGAMA@BERKELEY.EDU and **Somayeh Sojoudi** SOJOUDI@BERKELEY.EDU
Electrical Engineering and Computer Sciences Dept., University of California, Berkeley, CA 94709, USA

Abstract

The linear-quadratic controller is one of the fundamental problems in control theory. The optimal solution is a linear controller that requires access to the state of the entire system at any given time. When considering a network system, this renders the optimal controller a *centralized* one. The interconnected nature of a network system often demands a *distributed* controller, where different components of the system are controlled based only on local information. Unlike the classical centralized case, obtaining the optimal distributed controller is usually an intractable problem. Thus, we adopt a graph neural network (GNN) as a parametrization of distributed controllers. GNNs are naturally local and have distributed architectures, making them well suited for learning nonlinear distributed controllers. By casting the linear-quadratic problem as a self-supervised learning problem, we are able to find the best GNN-based distributed controller. We also derive sufficient conditions for the resulting closed-loop system to be stable. We run extensive simulations to study the performance of GNN-based distributed controllers and showcase that they are a computationally efficient parametrization with scalability and transferability capabilities.

Keywords: Distributed control, linear-quadratic control, graph neural networks

1. Introduction

Undoubtedly, linear dynamical systems are the cornerstone of countless information processing algorithms in a wide array of areas, including physics, mathematics, engineering and economics (Kailath, 1980; Rugh, 1996). Therefore, the ability to optimally control these systems is of paramount importance (Kwakernaak and Sivan, 1972; Bertsekas, 2005). Of particular interest is the case when linear systems are coupled with a quadratic cost, giving rise to the well-studied linear-quadratic control problem (Anderson and Moore, 1989; Makila and Toivonen, 1987; Dean et al., 2020). As it happens, the optimal linear-quadratic controller is *linear* and acts on the knowledge of the system state at a given time to produce the optimal control action for that time instant.

A special class of dynamical systems that has gained widespread attention are network systems. These systems are comprised of a set of interconnected components, capable of exchanging information and equipped with the ability to autonomously decide on an action to take. The objective is to coordinate the individual actions of the components so that they are conducive to the accomplishment of some global task (Bamieh et al., 2002; Nayyar et al., 2013; Fattahi et al., 2019c).

Linear network systems that seek to minimize a global quadratic cost can be easily controlled if we allow for a *centralized* approach. That is, if we assume that all components have instantaneous access to the state of all other components, they can readily compute the optimal control action. Furthermore, such an optimal centralized controller is linear. Computing optimal centralized controllers, however, face limitations in terms of scalability and implementation.

* This work was supported by grants from ONR, NSF and AFOSR.

The interconnected nature of network systems naturally imposes a distributed data structure. While this structure may affect the linear dynamics of the system, or even the objective quadratic cost, we are predominantly interested in leveraging the data structure to obtain *distributed* controllers. These are control actions that depend only on local information provided by components that share a connection and that can be computed separately by each component.

Imposing a distributed constraint on the linear-quadratic control problem renders it intractable in the most general case (Witsenhausen, 1968; Tsitsiklis and Athans, 1984). While there is a large class of distributed control problems that admit a convex formulation (Rotkowitz and Lall, 2006), many of them lead to complex solutions that do not scale with the size of the network (Tanaka and Parrilo, 2014; Fattahi et al., 2019a). An alternative approach is to adopt a linear parametrization of the controller and find a surrogate of the original problem that admits a scalable solution. The resulting controller is thus a sub-optimal linear distributed controller. Suboptimality guarantees are often obtained, and stability and robustness analysis of the resulting controller are provided (Fazelnia et al., 2017; Wang et al., 2019; Fattahi et al., 2019b)

However, even in the context of linear network system with a quadratic cost, the optimal distributed controller may not be linear (Witsenhausen, 1968). In this paper, we thus adopt a nonlinear parametrization of the controller. More specifically, we focus on the use of graph neural networks (GNNs) (Bronstein et al., 2017; Gama et al., 2020b). GNNs consist of a cascade of blocks (commonly known as layers) each of which applies a bank of graph filters followed by a pointwise nonlinearity (Bruna et al., 2014; Defferrard et al., 2016; Gama et al., 2019a). GNNs exhibit several desirable properties in the context of distributed control. Most importantly, they are naturally local and distributed, meaning that by adopting a GNN as a mapping between states and actions, a distributed controller is automatically obtained. Furthermore, they are permutation equivariant and Lipschitz continuous to changes in the network (Gama et al., 2020a). These two properties allow them to scale up and transfer to unknown networks (Ruiz et al., 2020).

Distributed controllers leveraging neural network techniques can be found in (Capella et al., 2003; Huang et al., 2005; Choy et al., 2006; Chen and Lin, 2013; Liu et al., 2015; Yang et al., 2017; Tolstaya et al., 2019; Li et al., 2020). These controllers typically use a distinct multi-layer perceptron (MLP) to parametrize the controller at each component (Capella et al., 2003; Huang et al., 2005; Choy et al., 2006; Chen and Lin, 2013; Liu et al., 2015; Yang et al., 2017), while GNNs are leveraged in (Tolstaya et al., 2019; Li et al., 2020) in the context of robotics problems.

The main contributions of this work are: (i) the use of graph neural networks (Section 3) as a practically useful nonlinear parametrization of the distributed controller for a linear-quadratic problem (Section 2), (ii) casting the linear-quadratic problem as a self-supervised learning problem that can be solved by traditional machine learning techniques [cf. (12)], (iii) a sufficient condition for the resulting GNN-based controller to stabilize the system (Proposition 1), and (iv) a comprehensive numerical simulation investigating the performance of GNN-based distributed controllers and its dependence on design hyperparameters, as well as its scalability and transferability (Section 4).

2. Distributed Linear-Quadratic Controllers

Consider a system of N components, each one described at time $t \in \{0, 1, 2, \ldots, T\}$ by a state vector $\mathbf{x}_i(t) \in \mathbb{R}^F$ for $i = 1, \ldots, N$. These components are also equipped with the ability to take an action $\mathbf{u}_i(t) \in \mathbb{R}^G$ that can influence future values of its own state, as well as the states of other components in the system, as determined by some given dynamic model . The state and the control

actions can be compactly described by matrices $\mathbf{X}(t) \in \mathbb{R}^{N \times F}$, $\mathbf{U}(t) \in \mathbb{R}^{N \times G}$, respectively, with

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}_1(t)^\mathsf{T} \\ \vdots \\ \mathbf{x}_N(t)^\mathsf{T} \end{bmatrix} \tag{1}$$

and analogously for $\mathbf{U}(t)$, i.e. the i^{th} row of $\mathbf{U}(t)$ is the action $\mathbf{u}_i(t) \in \mathbb{R}^G$ taken by component i at time t. We are particularly interested in linear system dynamics, modeled as (Kailath, 1980, Ch. 6)

$$\mathbf{X}(t+1) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}\mathbf{U}(t)\bar{\mathbf{B}} \tag{2}$$

with the given matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ known as the system matrix, and the matrices $\mathbf{B} \in \mathbb{R}^{N \times N}$ and $\bar{\mathbf{B}} \in \mathbb{R}^{G \times F}$ known as the control matrices. We note that F is the dimension of the state vector at each component and G is the dimension of the control action executed by each component. In this context, the matrix $\bar{\mathbf{B}}$ acts as a linear map between the G values of each individual control action $\mathbf{u}_i(t)$ and the F values of each individual state $\mathbf{x}_i(t)$.

A linear-quadratic regulator (LQR) is a controller that minimizes a quadratic cost on the states and the control actions (Bertsekas, 2005, Sec. 4.1):

$$\mathsf{J}\Big(\{\mathbf{X}(t)\}_{t=0}^{T}, \{\mathbf{U}(t)\}_{t=0}^{T-1}; \mathbf{Q}, \mathbf{R}\Big) = \big\|\mathbf{Q}^{1/2}\mathbf{X}(T)\big\|^2 + \sum_{t=0}^{T-1} \Big(\big\|\mathbf{Q}^{1/2}\mathbf{X}(t)\big\|^2 + \big\|\mathbf{R}^{1/2}\mathbf{U}(t)\big\|^2\Big) \tag{3}$$

for some given positive semidefinite matrices $\mathbf{Q}, \mathbf{R} \in \mathbb{R}^{N \times N}$ and for some given matrix norm $\|\cdot\|$. We note that if we set $F = G = 1$, then the state and control actions become vectors $\mathbf{x}(t), \mathbf{u}(t) \in \mathbb{R}^N$, respectively, and by choosing the Euclidean norm, the quadratic cost (3) results in the well-known cost of the traditional LQR problem (Anderson and Moore, 1989). In this case, the optimal control actions that solve the LQR problem can be computed directly by means of a linear map of the state value, i.e. $\mathbf{u}^\star(t) = \mathbf{K}_t^\star \mathbf{x}(t)$, with $\mathbf{K}_t^\star \in \mathbb{R}^{N \times N}$ having a closed-form solution in terms of $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$ (Bertsekas, 2005, Ch. 4). In summary, the objective of the LQR problem is to find a sequence of control actions $\{\mathbf{U}(t)\}$ such that (3) is minimized, subject to the dynamics in (2).

In what follows, we focus on the case where the system is distributed in nature. This means that the components of the system can only interact with those other components to which they have a direct connection. To describe this connectivity pattern, we model the system as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is the set of nodes with node v_i representing the i^{th} component in the system and where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges with $(v_i, v_j) \in \mathcal{E}$ if and only if the i^{th} component is connected to the j^{th} one. The distributed nature of the system typically has an impact on the structure of the system matrix $\mathbf{A}$ and the control matrix $\mathbf{B}$ [cf. (2)], usually respecting the topology of the graph (i.e. the sparsity of the adjacency matrix). More generally, we assume that the matrices $\mathbf{A}$ and $\mathbf{B}$ share the sparsity of some k-hop shortest path between neighbors, i.e. that the (i, j) entry of the matrix may be nonzero if and only if there is a k-hop path between v_i and v_j for some (unknown) value of k. We note that this may lead to non-sparse matrices, but that still exhibit a strong structure related to the underlying connectivity of the system. Examples include, both discrete-time (Gama and Ribeiro, 2019) and continuous-time (Olfati-Saber et al., 2007) diffusion models, as well as heat processes (Thanou et al., 2017), among others (Gama et al., 2019b).

Another fundamental aspect where the distributed nature of the system has a key impact is on the controller. More specifically, we concentrate on finding controllers that minimize the quadratic

cost (3) and whose action can be computed by means of operations that respect the connectivity of the system. We consider that the computational operations required to obtain each component's control action $\mathbf{u}_i(t)$ from the state $\mathbf{X}(t)$ only involve information relied by other components to which component i is connected. We call them *distributed* controllers and denote them by

$$\mathbf{U}(t) = \Phi\big(\mathbf{X}(t); \mathcal{G}\big) \tag{4}$$

where $\Phi : \mathbb{R}^{N \times F} \to \mathbb{R}^{N \times G}$ and where the mapping is explicitly parametrized by the graph $\mathcal{G}$. We have assumed, to ease the learning process, that the distributed controller is static.

The optimal distributed LQR controller can then be found by solving the following problem:

$$\min_{\Phi \in \mathbf{\Phi}_{\mathcal{G}}} \mathsf{J}\Big(\{\mathbf{X}(t)\}_{t=0}^{T}, \{\mathbf{U}(t)\}_{t=0}^{T-1}; \mathbf{Q}, \mathbf{R}\Big) \tag{5}$$

$$\text{s. t. } \mathbf{U}(t) = \Phi\big(\mathbf{X}(t); \mathcal{G}\big) \tag{6}$$

where $\mathbf{\Phi}_{\mathcal{G}}$ is the space of all possible mappings $\mathbb{R}^{N \times F} \to \mathbb{R}^{N \times G}$ that respect the structure of the graph. Solving problem (5)-(6) requires solving an optimization problem over the field of functions $\mathbf{\Phi}_{\mathcal{G}}$. This is mathematically intractable in the general case (Jahn, 2007), and requires specific approaches involving variational methods (Cassel, 2013), dynamic programming (Bertsekas, 2005) or kernel-based functions (Murphy, 2012, Ch. 14). However, when we drop the distributed constraint (6), problem (5) can actually be solved resulting in a linear controller; but this is a *centralized* controller that does not respect the distributed nature of the system. In any case, methods for solving (5)-(6) require large datasets and exhibit poor generalization (Goodfellow et al., 2016, Ch. 6).

Considering the inherent complexities of functional optimization, a popular approach is to adopt a specific model for the representation map Φ (Anthony and Bartlett, 1999, Ch. 2), leading to a parametric family of representations. Then, finding the best representation amounts to finding the optimal set of parameters, which results in a more tractable optimization problem over a finite-dimensional space (Engl et al., 1996, Ch. 9). One such parametrization is that of distributed linear controllers $\Phi(\mathbf{X}(t); \mathcal{G}) = \mathbf{H}(\mathcal{G})\mathbf{X}(t)\bar{\mathbf{H}}$, where we optimize over the space of all matrices $\mathbf{H}(\mathcal{G}) \in \mathbb{R}^{N \times N}$ (and $\bar{\mathbf{H}} \in \mathbb{R}^{F \times G}$) that respect the connectivity pattern of the system. Many properties of this parametric family of controllers have been studied, including stability, robustness and (sub)optimality (Fazelnia et al., 2017; Fattahi et al., 2019b).

However, it is known that even a linear system like (2) may have a nonlinear optimal controller if we force a distributed nature on it (Witsenhausen, 1968). This suggests that it would be more convenient to work with nonlinear parametrizations, rather than linear ones. In particular, we focus on graph neural networks (GNNs) (Bruna et al., 2014; Defferrard et al., 2016; Gama et al., 2019a) as they are nonlinear mappings that exhibit several desirable properties. Fundamentally, they are naturally computed by means of local and distributed operations. This implies that any controller that is parametrized by means of a GNN respects the distributed nature of the system (as given by the graph $\mathcal{G}$), naturally incorporating the distributed constraint into the chosen parametrization.

3. Graph Neural Networks

Graph signal processing (GSP) is a convenient framework to describe distributed problems (Sandryhaila and Moura, 2013; Shuman et al., 2013). For a given graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, we define a *graph signal* as the mapping $\mathsf{x} : \mathcal{V} \to \mathbb{R}^F$ that assigns an F-dimensional vector to each node, $\mathsf{x}(v_i) = \mathbf{x}_i \in$

$\mathbb{R}^F$. We can thus conveniently describe a graph signal by means of a matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ so that its i^{th} row corresponds to the value of the signal at that node, $\mathsf{x}(v_i) = \mathbf{x}_i$. It is immediate that we can consider both the system state $\mathbf{X}(t)$ and the control action $\mathbf{U}(t)$ as time-varying graph signals, $\mathsf{x}_t : \mathcal{V} \to \mathbb{R}^F$ and $\mathsf{u}_t : \mathcal{V} \to \mathbb{R}^G$, respectively [cf. (1), (2)].

Describing a graph signal by means of a matrix $\mathbf{X}$ is mathematically convenient but, in doing so, we rescind the relationship between the signal and the underlying graph support. In other words, the matrix $\mathbf{X}$ contains no information about the graph. To recover this relationship, we start by defining the *support matrix* $\mathbf{S} \in \mathbb{R}^{N \times N}$, which respects the sparsity pattern of the graph, i.e. $[\mathbf{S}]_{ij} = 0$ for $i \neq j$ whenever $(v_j, v_i) \notin \mathcal{E}$. Therefore, the matrix $\mathbf{S}$ represents the underlying graph support, and examples in the literature include the adjacency (Sandryhaila and Moura, 2013) and the Laplacian (Shuman et al., 2013) matrices, as well as their normalized counterparts (Defferrard et al., 2016).

The support matrix $\mathbf{S}$ can then be used to define a linear mapping such that the output $\mathbf{SX}$ is another graph signal whose values are related to the underlying graph support. More specifically, the f^{th} output value at the i^{th} component is given by

$$[\mathbf{SX}]_{if} = \sum_{j=1}^{N} [\mathbf{S}]_{ij} [\mathbf{X}]_{jf} = \sum_{j:v_j \in \mathcal{N}_i} [\mathbf{S}]_{ij} [\mathbf{X}]_{jf} \tag{7}$$

where $\mathcal{N}_i = \{v_j : (v_j, v_i) \in \mathcal{E}\} \cup \{v_i\}$ is the neighborhood of node v_i. Note that while the first equality corresponds to the definition of the matrix multiplication, the second equality holds because of the sparsity pattern of $\mathbf{S}$, i.e. the only nonzero entries in $\mathbf{S}$ correspond to those nodes that share an edge. In short, the linear map (7) yields an output graph signal that is a linear combination of neighboring values of the input graph signal.

The support matrix $\mathbf{S}$ acts as an elementary operator between graph signals. More precisely, it is a proper generalization of the unit time-shift (or time-delay) operator in traditional signal processing; it just shifts the signal through the graph (diffuses the signal), often receiving the name of *graph shift operator* (GSO). Therefore, we can use $\mathbf{S}$ as the basic building block to construct linear graph filters as follows (Sandryhaila and Moura, 2013):

$$\mathsf{H}(\mathbf{X}; \mathbf{S}) = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k \tag{8}$$

for some polynomial order K and where $\mathbf{H}_k \in \mathbb{R}^{F \times G}$ is the corresponding *filter tap* (or filter coefficient). Certainly, the output of the graph filtering operation (8) is another graph signal, but with G values at each node, so that $\mathsf{H} : \mathbb{R}^{N \times F} \to \mathbb{R}^{N \times G}$ is a linear map between graph signals.

Linear graph filters as in (8) are local and distributed operations. To understand this, note that matrix multiplications to the left of $\mathbf{X}$ compute linear combinations of signal values across different nodes, while matrix multiplications to the right of $\mathbf{X}$ compute linear combinations of signal values within the same node. Therefore, for the operation to be local, multiplications on the right need to respect the sparsity of the graph, i.e. only combine values of nodes that are connected to each other (multiplications to the left can be arbitrary). This is precisely the case when multiplying by $\mathbf{S}^k = \mathbf{S}(\mathbf{S}^{k-1})$ since the operation can be computed by k repeated exchanges with one-hop neighbors. Additionally, by storing the values of the filter taps $\{\mathbf{H}_k\}$, each node can compute the corresponding output separately by leveraging the information provided by the k exchanges with its one-hop neighbors. Thus, a linear graph filter is a local and distributed operation, making them well suited for learning linear distributed controllers.

The graph filter (8) is a proper generalization of the convolution operation and, thus, is usually referred to as a *graph convolution* as well (Gama et al., 2020b). Technically speaking, (8) is a finite-impulse response (FIR) graph filter. However, due to the finite nature of graphs, it also encompasses infinite-impulse response (IIR) graph filters. In what follows, we focus on filters of the form (8) and generically refer to them as graph filters.

Graph filters (8) naturally model linear distributed controllers, i.e. $\mathbf{U}(t) = \mathsf{H}(\mathbf{X}; \mathbf{S})$. However, we are interested in learning distributed controllers that are capable of capturing nonlinear relationships between the state and the control action. Towards this end, we introduce GNNs (Gama et al., 2020b), which cascade L layers, each applying a graph filter followed by a pointwise nonlinearity

$$\Phi(\mathbf{X}; \mathbf{S}) = \mathbf{X}_L \quad , \quad \mathbf{X}_\ell = \sigma\Big(\mathsf{H}_\ell(\mathbf{X}_{\ell-1}; \mathbf{S})\Big) \tag{9}$$

for $\ell = 1, \ldots, L$, with $\mathbf{X}_0 = \mathbf{X}$ being the input graph signal. The function $\sigma : \mathbb{R} \to \mathbb{R}$ is a pointwise nonlinearity that acts on each entry of the graph signal (in a slight abuse of notation, we write $\sigma(\mathbf{X})$ to denote $[\sigma(\mathbf{X})]_{ij} = \sigma([\mathbf{X}]_{ij})$). The output of each layer is a graph signal $\mathbf{X}_\ell \in \mathbb{R}^{N \times F_\ell}$ with $F_0 = F$ and $F_L = G$. The specific nonlinearity σ to be used, the size of the graph signals F_ℓ, and the number of filters taps K_ℓ are design choices. We remark that once these values are chosen, the space of possible GNN-based controllers is completely characterized by the values of the filters taps $\{\mathbf{H}_{\ell k}\}_{\ell,k}$ at each layer (Gama et al., 2020b).

The GNN (9) exhibits several desirable properties for learning distributed nonlinear controllers. Fundamentally, due to the pointwise nature of the nonlinearity, they retain the local and distributed nature of graph filters. This means that their output can be computed separately at each node, by exchanging information with one-hop neighbors only. Additionally, they are permutation equivariant and Lipschitz continuous to changes in the graph support $\mathbf{S}$ (Gama et al., 2020a). These two properties show how the GNNs effectively exploit the graph structure of the system to improve learning (and thus, they are expected to work better when the dynamics are also graph-dependent), and facilitate scalability and transferability. Finally, we note that while permutation equivariance and Lipschitz continuity to perturbations are properties also exhibited by linear graph filters, the GNNs leverage the nonlinearity to increase the discriminative power, helping to capture more information that graph filters do (Pfrommer et al., 2020).

All GNN-based controllers are naturally distributed. However, we are interested in those that exhibit a small quadratic cost. That is, we are interested in finding the appropriate filter taps such that (3) is minimized

$$\min_{\{\mathbf{H}_{\ell k}\}_{\ell,k}} \mathsf{J}\Big(\{\mathbf{X}(t)\}_{t=0}^{T}, \{\mathbf{U}(t)\}_{t=0}^{T-1}; \mathbf{Q}, \mathbf{R}\Big) \tag{10}$$

$$\text{s. t. } \mathbf{U}(t) = \Phi\big(\mathbf{X}(t); \mathbf{S}\big). \tag{11}$$

Note that problem (10)-(11) is a finite-dimensional optimization one that has $\sum_{\ell=1}^{L} F_\ell F_{\ell-1} K_\ell$ dimensions (independent of the size of the system N). Note, also, that the distributed constraint (6) has been incorporated by forcing Φ to be a GNN (9) [cf. (11)].

Problem (10)-(11) is nonconvex due to the GNN-based controller constraint (11). Thus, to approximately solve this problem, we leverage an empirical risk minimization (ERM) approach that is typical in learning (Vapnik, 2000). To do this, we create a *training set* $\mathcal{T} = \{\mathbf{X}_{1,0}, \ldots, \mathbf{X}_{|\mathcal{T}|,0}\}$ containing $|\mathcal{T}|$ samples $\mathbf{X}_{p,0}$ drawn independently from some distribution p, which we consider to

be different random initializations of the system. We then focus on the ERM problem given by

$$\begin{aligned} \min_{\{\mathbf{H}_{\ell k}\}_{\ell,k}} \; & \sum_{p=1}^{|\mathcal{T}|} \mathsf{J}\Big(\{\mathbf{X}_p(t)\}_{t=0}^{T}, \{\mathbf{U}_p(t)\}_{t=0}^{T-1}; \mathbf{Q}, \mathbf{R}\Big) \\ \text{s. t. } & \mathbf{U}_p(t) = \mathbf{\Phi}\big(\mathbf{X}_p(t); \mathbf{S}\big) \\ & \mathbf{X}_p(t+1) = \mathbf{A}\mathbf{X}_p(t) + \mathbf{B}\mathbf{U}_p(t)\bar{\mathbf{B}} \;,\; \mathbf{X}_p(0) = \mathbf{X}_{p,0}. \end{aligned} \tag{12}$$

Recall that $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$ are given by the problem, and $\mathbf{S}$ is an appropriately chosen support matrix (i.e. adjacency, Laplacian, etc.). We solve (12) by means of an algorithm based on stochastic gradient descent (Kingma and Ba, 2015), efficiently computing the gradient of J with respect to the parameters $\mathbf{H}_{\ell k}$ by means of the back-propagation algorithm (Rumelhart et al., 1986). To estimate the performance of the *learned* controllers (i.e. those obtained by solving (12)), we generate a new set of initial states, called the test set, and compute the average quadratic cost (3) on the resulting trajectories. In essence, we transform the optimization problem (10)-(11) into a *self-supervised* ERM problem (12) that is solved through simulated data.

Henceforth, we focus on a two-layer GNN-based controller given by [cf. (9)]

$$\mathbf{U}(t) = \mathsf{H}_2\Big(\sigma\big(\mathsf{H}_1(\mathbf{X}(t); \mathbf{S})\big); \mathbf{S}\Big). \tag{13}$$

We do so because, in practice, each communication exchange between one-hop neighbors takes time. Thus, having many layers and/or large values of K_ℓ may require unrealistically fast communications or would only be applicable to particularly slow processes. For more details on time-varying graph signals and communication delays, please refer to Isufi et al. (2019); Gama et al. (2020b,c).

We now give a sufficient condition for a GNN-based learned controller to stabilize the system in the input-state sense of (**?**).

Proposition 1 *Consider a linear dynamical system* (2) *controlled by* $\mathbf{U}(t) = \mathbf{\Phi}(\mathbf{X}(t)) + \mathbf{E}(t)$ *with* $\mathbf{\Phi}$ *the GNN-based controller given by* (13) *and with a nonlinearity* σ *such that* $|\sigma(x)| \leq |x|$ *for all* $x \in \mathbb{R}$*; and where* $\mathbf{E}(t)$ *is a disturbance term or exploratory signal that satisfies* $\sum_{t=0}^{\infty} \|\mathbf{E}(t)\| < \infty$. *Then, the closed-loop system is input-state stable, i.e there exist constants* $\beta_0, \beta_1 \geq 0$ *such that*

$$\sum_{t=0}^{\infty} \|\mathbf{X}(t)\| \leq \beta_0 + \beta_1 \sum_{t=0}^{\infty} \|\mathbf{E}(t)\| \tag{14}$$

as long as H_1 *and* H_2 *satisfy*

$$b c_2 c_{1a} + c_2 \bar{c}_{1b} < \big(1 - a\big)\big(1 - c_2 \bar{c}_{1b}\big) \tag{15}$$

where

$$a = \|\mathbf{A}\| \;,\; b = \|\mathbf{B}\| \;,\; c_2 = \sum_{g=1}^{G} c_2^g \;,\; c_{1a} = \sum_{g=1}^{F_1} c_{1a}^g \;,\; \bar{c}_{1b} = \sum_{g=1}^{F_1} \bar{c}_{1b}^g \tag{16}$$

for $\|\cdot\|$ *is the spectral norm, and with* $\mathbf{H}_\ell^{fg}(\mathbf{S}) = \sum_{k=0}^{K_\ell} [\mathbf{H}_{\ell k}]_{fg} \mathbf{S}^k$ *being a polynomial built with the* (f, g) *entries of the filter taps* $\{\mathbf{H}_k\}$, $\bar{\mathbf{H}}_{1k} = \bar{\mathbf{B}}\mathbf{H}_{1k}$ *and*

$$c_2^g = \max_{f=1,\ldots,F_1} \|\mathbf{H}_2^{fg}(\mathbf{S})\| \;,\; c_{1a}^g = \max_{f=1,\ldots,F} \|\mathbf{H}_1^{fg}(\mathbf{S})\mathbf{A}\| \;\; \bar{c}_{1b}^g = \max_{f=1,\ldots,G} \|\bar{\mathbf{H}}_1^{fg}(\mathbf{S})\mathbf{B}\|. \tag{17}$$

Proof See Supplementary Material in Gama and Sojoudi (2020). ■

Proposition 1 is a sufficient condition for the closed-loop system to be input-state stable. We note that the condition on the nonlinearity is mild and is satisfied by most popular nonlinearities (ReLUs, $\tanh$, sigmoid, etc.). We remark that (15) is a conservative bound, in that systems that do not satisfy it may be input-state stable as well.

4. Numerical Experiments

We showcase the performance of GNNs for learning decentralized controllers in the linear-quadratic problem. We consider the traditional finite-time horizon formulation,

$$\begin{aligned} \min \quad & \mathbf{x}(T)^{\mathsf{T}}\mathbf{Q}\mathbf{x}(T) + \sum_{t=0}^{T-1} \Big(\mathbf{x}(t)^{\mathsf{T}}\mathbf{Q}\mathbf{x}(t) + \mathbf{u}(t)^{\mathsf{T}}\mathbf{R}\mathbf{u}(t) \Big) \\ \text{subject to } & \mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \ , \ t = 0, 1, \ldots, T-1 \end{aligned} \tag{18}$$

for some given matrices $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R} \in \mathbb{R}^{N \times N}$ with $\mathbf{Q}, \mathbf{R}$ being positive definite. We note that $\mathbf{x}(t), \mathbf{u}(t) \in \mathbb{R}^N$ so that (18) is a particular case of (2)-(3) with $F = G = 1$, where $\bar{\mathbf{B}} = \bar{b}$ has been absorbed in $\mathbf{B}$, and where we use the ℓ_2 norm in (3). In short, we consider the state and the control action of each node to be a scalar, $[\mathbf{x}(t)]_i = x_i(t)$ and $[\mathbf{u}(t)]_i = u_i(t)$, respectively.

Setting. We consider a system with N nodes placed uniformly at random in a plane, and we build the corresponding graph $\mathcal{G}$ by keeping only the 5-nearest neighbors of each node. We adopt a support matrix $\mathbf{S}$ given by the adjacency matrix and normalized by the largest eigenvalue $\|\mathbf{S}\|_2 = 1$. The system matrices $\mathbf{A}$ and $\mathbf{B}$ are chosen at random and we set $\mathbf{Q} = \mathbf{R} = \mathbf{I}$. Unless otherwise specified, we set $N = 20$ nodes, $T = 50$ instances, and the matrices $\mathbf{A}$ and $\mathbf{B}$ are made to share the same eigenvector basis as $\mathbf{S}$, while the eigenvalues are chosen at random using a zero-mean unit-variance Gaussian, and are then normalized to have $\|\mathbf{A}\|_2 = 0.995$ and $\|\mathbf{B}\|_2 = 1$.

Controllers. We consider 5 different controllers. (i: Optim) As a baseline, we use the optimal, linear and centralized controller, which can be computed recursively by knowing $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$. (ii: MLP) We consider a *learned* centralized controller given by a two-layer fully-connected neural network with $F_{\text{MLP}}N$ hidden units in the first layer, N output units, and a σ_{MLP} nonlinearity. (iii: D-MLP) The decentralized controller in (Huang et al., 2005) which assigns an individual two-layer fully connected neural network to each node with $\sigma_{\text{D-MLP}}$ nonlinearity. (iv: GF) A decentralized linear graph filter bank controller [cf. (8)] consisting of a cascade of two banks with $F_0 = F_2 = 1, F_1 = F_{\text{GF}}$ and $K_1 = K_{\text{GF}}, K_2 = 0$. (v: GNN) A two-layer GNN [cf. (9), (13)] with $F_0 = F_2 = 1, F_1 = F_{\text{GF}}$, and $K_1 = K_{\text{GF}}, K_2 = 0$, and a σ_{GNN} nonlinearity. Unless otherwise specified, all nonlinearities are $\tanh$, and we set $F_{\text{MLP}} = F_{\text{GNN}} = 32$, $F_{\text{GF}} = 16$ and $K_{\text{GF}} = K_{\text{GNN}} = 3$.

Training and evaluation. For training, we consider the ERM equivalent problem of (18) [cf. (10)-(11), (12)] which we minimize over a training set consisting of $|\mathcal{T}|$ initial states $\mathbf{x}_{p,0}$, $p = 1, \ldots, |\mathcal{T}|$, with entries randomly distributed following a zero-mean unit variance Gaussian. We adopt an ADAM optimizer with learning rate μ and forgetting factors 0.9 and 0.999 (Kingma and Ba, 2015). The training procedure consists of 30 epochs with a batch size of 20. Moreover, every 5 training steps we run an evaluation over a validation set consisting of 50 samples. After the training is

Table 1: Average (std. deviation) normalized cost for different hyperparameters. Cost of other controllers: (i: Optim) $0.9961(\pm 0.0007)$, (ii: MLP) $0.999(\pm 0.002)$, (iii: D-MLP) $1.11(\pm 0.02)$.

$F_{\text{GF}}/K_{\text{GF}}$	1	2	3
16	$10(\pm 10)$	$1.5(\pm 0.5)$	$\mathbf{1.21(\pm 0.05)}$
32	$1.6(\pm 0.8)$	$1.3(\pm 0.2)$	$1.3(\pm 0.2)$
64	$1.5(\pm 0.3)$	$1.6(\pm 1.0)$	$2(\pm 1)$

$F_{\text{GNN}}/K_{\text{GNN}}$	1	2	3
16	$1.26(\pm 0.07)$	$1.20(\pm 0.05)$	$1.18(\pm 0.04)$
32	$1.26(\pm 0.08)$	$1.19(\pm 0.05)$	$\mathbf{1.17(\pm 0.05)}$
64	$1.26(\pm 0.07)$	$1.20(\pm 0.05)$	$1.18(\pm 0.05)$

finished, we retain the model parameters that have resulted in the lowest validation cost. For evaluation, we create a test set of 50 samples in analogous fashion and run the resulting controllers. For each controller, we compute the cost given by (18) for each trajectory, and average over all 50 trajectories. For a fair comparison, we normalize the cost by the lower bound for decentralized controllers provided in (Fazelnia et al., 2017). To account for the randomness in the data generation, for each experiment we run 10 different graph realizations, and we run 10 different instances of matrices $\mathbf{A}, \mathbf{B}$ for each graph realization. The reported results include the average over these realizations as well as the estimated standard deviation. Unless otherwise specified, we set $|\mathcal{T}| = 500$ and $\mu = 0.01$ (except for training MLP where we set $\mu = 0.001$).

Experiment 1: Design hyperparameters. In the first experiment, we study the dependency of (iv: GF) and (v: GNN) with the number of features F_{GF}, F_{GNN} and number of filter taps K_{GF}, K_{GNN}. Results are shown in Table 1. First, it can be observed that for the GF controller, the behavior with F_{GF} and K_{GF} is erratic, as evidenced not only by the different average costs, but also by the larger standard deviations. Notice that considering information from farther away neighbors (increasing K_{GF}) improves performance. The performance of the GNN controller, on the other hand, is more consistent and exhibits relatively good results in all cases. This does not appear to be affected by the number of chosen features F_{GNN}, but it improves with increasing the neighborhood information K_{GNN}. We have also ran experiments for different values of learning rate $\mu \in \{0.001, 0.005, 0.01\}$, and observed that this hyperparameter considerably impacts the performance of the learned controllers. In particular, the GF controller fails for almost all values of F_{GF} and K_{GF} when $\mu \in \{0.001, 0.005\}$. The GNN controller, on the other hand, succeeds for other values of μ but at a slightly higer cost ($1.21(\pm 0.05)$ for $\mu = 0.001$ and $1.18(\pm 0.05)$ for $\mu = 0.005$). From this experiment, we adopt the values of $F_{\text{GF}} = 16, F_{\text{GNN}} = 32, K_{\text{GF}} = K_{\text{GNN}} = 3$ and $\mu = 0.01$ to be used from now on.

Experiment 2: Comparison between controllers. In the above experiment, we also computed the optimal centralized controller (i: Optim), a learnable centralized controller (ii: MLP) and a learnable decentralized controller (iii: D-MLP). We ran the methods for different values of μ and F_{MLP} and kept the ones with the best performance, namely $\mu = 0.001$ and $F_{\text{MLP}} = 32$ for MLP and $\mu = 0.01$ for D-MLP. The normalized costs obtained are $0.9961(\pm 0.0007)$ for Optim, $0.999(\pm 0.002)$ for MLP and $1.11(\pm 0.02)$ for D-MLP. We note that the centralized controllers perform better, as expected, and, in fact, they have a lower cost than the lower bound (Fazelnia et al., 2017). This is expected since the space of centralized controllers contains the space of decentralized ones, and thus any optimal centralized controller is bound to be at least as good as any optimal decentralized one. Next, we observe that D-MLP has a lower cost than GF and GNN. This is also expected since the representation space of D-MLP contains that of GNNs. However, as we see in Experiment 4, this controller does not scale, since the optimization space grows with the size of the network, making it increasingly difficult to navigate during the training process.

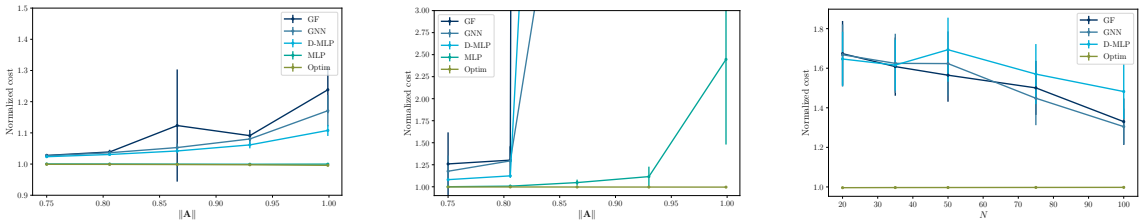

Figure 1. (Left) Change in cost for structured matrix $\mathbf{A}$. (Middle) Change in cost for random matrix $\mathbf{A}$. The higher the value of $\|\mathbf{A}\|$ the harder it is to control the system. We see that when the system matrix shares the same structure as the graph (i.e. same eigenvectors), the controllers are better at learning how to handle the system. We see that, when the structure is random, all decentralized controllers fail to learn appropriate actions when $\|\mathbf{A}\| > 0.8$. (Right) Change in cost for increasingly bigger systems. We observe that the D-MLP controller does not transfer at scale since, for bigger systems, it yields a higher cost than the GNN-based controller.

Experiment 3: Dependence on the system matrix A. In this experiment, we analyze the ability of the learned controller to successfully handle different system matrices $\mathbf{A}$. We consider, first, the case when $\mathbf{A}$ and $\mathbf{S}$ have the same set of eigenvectors; and, second, when $\mathbf{A}$ is completely random and bears no relationship with the underlying graph support. Additionally, we consider different values of $\|\mathbf{A}\|$ where we recall that the larger the value of $\|\mathbf{A}\|$ is, the harder the system is to control. The results are shown in Fig. 1. First, it is evident that when the system dynamics share the same structure as the underlying graph support, the decentralized learning methods are capable of capturing this structure to improve their performance. Even as $\|\mathbf{A}\|$ gets closer to 1 and the system becomes less stable [cf. Prop. 1], the learned controllers work well. Second, when the system matrix $\mathbf{A}$ is completely arbitrary, then the decentralized controllers fail.

Experiment 4: Transferability and scalability. In the last experiment, we consider the case in which we train the learnable controllers in a graph with $N = 20$ nodes, but then we test it on increasingly bigger systems of $N \in \{35, 50, 75, 100\}$ nodes. The results show in Fig. 1 illustrate that, while the D-MLP performs better when tested in a small system, it does not transfer to larger systems. This is because it assigns a different fully connected neural network controller to each component. Thus, when tested on larger systems, it has to replicate this controller on other components and that may have a substantially different topological neighborhood. GNN-based controller, on the other hand, successfully adapt to larger systems, even when trained on small ones.

5. Conclusions

Finding optimal distributed controllers is intractable in the most general case, and even in specific cases where the problem admits a convex formulation, the resulting controller does not scale up. Since it is expected that the optimal distributed controller is a nonlinear function of the states, we proposed to adopt a GNN to parametrize the controller. GNNs are well suited since they are nonlinear, naturally distributed, computationally efficient, scale and transfer. We observed in numerical experiments that they exhibit good performance.

This preliminary investigation of GNN-based controllers shows their potential for distributed control and opens up several avenues of future research. First, the sufficient condition for closed-loop stability is conservative and can be improved. Second, we can analyze the robustness of the GNN-based controller to changes in the system connectivity as well as in the matrix that describe the linear dynamics. Third, we can investigate the suboptimality of the GNN-based controllers.

References

B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Ser. Inform. Syst, Sci. Prentice-Hall, Englewood Cliffs, NJ, 1989.

M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, Cambridge, UK, 1999.

B. Bamieh, F. Paganini, and M. A. Dahleh. Distributed control of spatially invariant systems. *IEEE Trans. Autom. Control*, 47(3):1091–1107, July 2002.

D. P. Bertsekas. *Dynamic Programming and Optimal Control: Vol. I*. Ser. Optimization Comput. Athena Scientific, Belmont, MA, 3rd edition, 2005.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, July 2017.

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd Int. Conf. Learning Representations*, pages 1–14, Banff, AB, 14-16 Apr. 2014.

J. V. Capella, A. Bonastre, and R. Ors. An advanced and distributed control architecture based on intelligent agents and neural networks. In *IEEE Int. Workshop Intell. Data Acquisition Advanced Computing Syst.: Technol. Appl.*, pages 278–283, Lviv, Ukraine, 8-10 Sep. 2003. IEEE.

K. W. Cassel. *Variational Methods with Applications in Science and Engineering*. Cambridge University Press, New York, NY, 2013.

S.-Y. Chen and F.-J. Lin. Decentralized pid neural network control for five degree-of-freedom active magnetic bearing. *Eng. Appl. Artificial Intell.*, 26(3):962–973, March 2013.

M. C. Choy, D. Srinivasan, and R. L. Cheu. Neural networks for continuous online control. *IEEE Trans. Neural Netw.*, 17(6):1511–1531, Nov. 2006.

S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu. On the sample complexity of the linear quadratic regulator. *Found. Comput. Math.*, 20:633–679, 2020.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *30th Conf. Neural Inform. Process. Syst.*, pages 3844–3858, Barcelona, Spain, 5-10 Dec. 2016. Neural Inform. Process. Foundation.

H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*, volume 375 of *Ser. Math. Its Appl.* Kluwer Academic Publishers, Dordrecht, the Netherlands, 1996.

S. Fattahi, G. Fazelnia, J. Lavaei, and M. Arcak. Transformation of optimal centralized controllers into near-globally optimal static distributed controllers. *IEEE Trans. Autom. Control*, 64(1):66–80, Jan. 2019a.

S. Fattahi, N. Matni, and S. Sojoudi. Efficient learning of distributed linear-quadratic controllers. *arXiv:1909.09895v2 [math.OC]*, 11 Oct. 2019b. URL http://arxiv.org/abs/1909.09895.

S. Fattahi, N. Matni, and S. Sojoudi. Learning sparse dynamical systems from a single sample trajectory. In *58th IEEE Conf. Decision, Control*, pages 2683–2689, Nice, France, 11-13 Dec. 2019c. IEEE.

G. Fazelnia, R. Madani, A. Kalbat, and J. Lavaei. Convex relaxation for optimal distributed control problems. *IEEE Trans. Autom. Control*, 62(1):206–221, Jan. 2017.

F. Gama and A. Ribeiro. Ergodicity in stationary graph processes: A weak law of large numbers. *IEEE Trans. Signal Process.*, 67(10):2761–2774, 15 May 2019.

F. Gama and S. Sojoudi. Graph neural networks for distributed linear-quadratic control. *arXiv:2011.05360v2 [eess.SY]*, 13 Nov. 2020. URL http://arxiv.org/abs/2011.05360.

F. Gama, A. G. Marques, G. Leus, and A. Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Trans. Signal Process.*, 67(4):1034–1049, 15 Feb. 2019a.

F. Gama, E. Isufi, A. Ribeiro, and G. Leus. Controllability of bandlimited graph processes over random time varying graphs. *IEEE Trans. Signal Process.*, 67(24):6440–6454, 15 Dec. 2019b.

F. Gama, J. Bruna, and A. Ribeiro. Stability properties of graph neural networks. *IEEE Trans. Signal Process.*, 68:5680–5695, 25 Sep. 2020a.

F. Gama, E. Isufi, G. Leus, and A. Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Process. Mag.*, 37(6):128–138, Nov. 2020b.

F. Gama, E. Tolstaya, and A. Ribeiro. Graph neural networks for decentralized controllers. *arXiv:2003.10280v2 [cs.LG]*, 21 Oct. 2020c. URL http://arxiv.org/abs/2003.10280.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Ser. Adaptive Comput. Mach. Learning. The MIT Press, Cambridge, MA, 2016.

S. N. Huang, K. K. Tan, and T. H. Lee. Decentralized control of a class of large-scale nonlinear systems using neural networks. *Automatica*, 41(9):1645–1649, Sep. 2005.

E. Isufi, A. Loukas, N. Perraudin, and G. Leus. Forecasting time series with VARMA recursions on graphs. *IEEE Trans. Signal Process.*, 67(18):4870–4885, 15 Sep. 2019.

J. Jahn. *Introduction ot the Theory of Nonlinear Optimization*. Springer-Verlag, Berlin, Germany, 3rd edition, 2007.

T. Kailath. *Linear Systems*. Ser. Inform. Syst, Sci. Prentice-Hall, Englewood Cliffs, NJ, 1980.

D. P. Kingma and J. L. Ba. ADAM: A method for stochastic optimization. In *3rd Int. Conf. Learning Representations*, pages 1–15, San Diego, CA, 7-9 May 2015.

H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. John Wiley & Sons, New York, NY, 1972.

Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, Las Vegas, NV, 25-29 Oct. 2020. IEEE.

D. Liu, C. Li, H. Li, D. Wang, and H. Ma. Neural-network-based decentralized control of continuous-time nonlinear interconnected systems with unknown dynamics. *Neurocomputing*, 165:90–98, 1 Oct. 2015.

P. Makila and H. Toivonen. Computational methods for parametric LQ problems–A survey. *IEEE Trans. Autom. Control*, 32(8):658–671, Aug. 1987.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Ser. Adaptive Comput. Mach. Learning. The MIT Press, Cambridge, MA, 2012.

A. Nayyar, A. Mahajan, and D. Teneketzis. Decentralized stochastic control with partial history sharing: A common information approach. *IEEE Trans. Autom. Control*, 58(7):1644–1658, July 2013.

R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proc. IEEE*, 95(1):215–233, Jan. 2007.

S. Pfrommer, F. Gama, and A. Ribeiro. Discriminability of single-layer graph neural networks. *arXiv:2010.08847v2 [eess.SP]*, 21 Oct. 2020. URL http://arxiv.org/abs/2010.08847.

M. Rotkowitz and S. Lall. A characterization of convex problems in decentralized control. *IEEE Trans. Autom. Control*, 51(2):274–286, Feb. 2006.

W. J. Rugh. *Linear Systems Theory*. Ser. Inform. Syst, Sci. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1996.

L. Ruiz, L. F. O. Chamon, and A. Ribeiro. Graphon neural networks and the transferability of graph neural networks. *arXiv:2006.03548v2 [cs.LG]*, 20 Oct. 2020. URL http://arxiv.org/abs/2006.03548.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986.

A. Sandryhaila and J. M. F. Moura. Discrete signal processing on graphs. *IEEE Trans. Signal Process.*, 61(7):1644–1656, 1 Apr. 2013.

D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Process. Mag.*, 30(3):83–98, May 2013.

T. Tanaka and P. A. Parrilo. Optimal output feedback architecture for triangular lqg problems. In *2014 Amer. Control Conf.*, pages 5730–5735, Portland, OR, 6-4 June 2014. IEEE.

D. Thanou, X. Dong, D. Kressner, and P. Frossard. Learning heat diffusion graphs. *IEEE Trans. Signal, Inform. Process. Networks*, 3(3):484–499, Sep. 2017.

E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro. Learning decentralized controllers for robot swarms with graph neural networks. In *Conf. Robot Learning 2019*, volume 100, pages 671–682, Osaka, Japan, 30 Oct.-1 Nov. 2019. Proc. Mach. Learning Res.

J. N. Tsitsiklis and M. Athans. One the complexity of decentralized decision making and detection problems. In *23rd IEEE Conf. Decision, Control*, pages 1638–1641, Las Vegas, NV, 12-14 Dec. 1984. IEEE.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Ser. Statist. Eng. Inform. Sci. Springer-Verlag, New York, NY, 2nd edition, 2000.

Y.-S. Wang, N. Matni, and J. C. Doyle. A system-level approach to controller synthesis. *IEEE Trans. Autom. Control*, 64(10):4079–4093, Oct. 2019.

H. S. Witsenhausen. A counterexample in stochastic optimum control. *SIAM J. Control*, 6(1): 131–147, 1968.

S. Yang, Y. Cao, Z. Peng, G. Wen, and K. Guo. Distributed formation control of nonholonomic autonomous vehicle via RBF neural network. *Mech. Syst. Signal Process.*, 87(B):81–95, 15 March 2017.