Systems & Control Letters 62 (2013) 85–96

Contents lists available at SciVerse ScienceDirect

Systems & Control Letters

journal homepage: www.elsevier.com/locate/sysconle

Performance and design of cycles in consensus networks☆

Daniel Zelazo[a,*], Simone Schuler[b], Frank Allgöwer[b]

[a] Faculty of Aerospace Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel

[b] University of Stuttgart, Institute for Systems Theory & Automatic Control, Pfaffenwaldring 9, 70550 Stuttgart, Germany

ARTICLE INFO

Article history:
Received 18 December 2011
Received in revised form
23 October 2012
Accepted 28 October 2012
Available online 7 December 2012

Keywords:
Consensus protocol
Performance
Graph theory
SDP

ABSTRACT

This work explores the role of cycles in consensus seeking networks for analysis and synthesis purposes. Cycles are critical for many reasons including improving the convergence rate of the system, resilience to link failures, and the overall performance of the system. The focus of this work examines how cycles impact the $\mathcal{H}_2$ performance of consensus networks. A first contribution shows that the addition of cycles always improves the performance of the system. We provide an analytic characterization of how the addition of edges improves the performance, and show that it is related to the inverse of the cycle lengths and the number of shared edges between independent cycles. These results are then used to consider the design of consensus networks. In this direction we present an ℓ_1-relaxation method that leads to a convex program for adding a fixed number of edges to a consensus networks. We also demonstrate how this relaxation can be used to embed additional performance criteria, such as maximization of the algebraic connectivity of the graph.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

The consensus protocol has recently emerged as a canonical model for studying networked dynamic systems. The simplicity of the model, most often presented as a collection of single integrators interacting over a communication graph, reveals a deep connection between its dynamic behavior and the underlying properties of the graph [1]. The use of consensus models, even beyond its analytical elegance, is its practical relevance in applications ranging from optimization and sensor fusion to problems in formation control and distributed control and estimation [2–6]. It is precisely the utility and simplicity of this model that has pushed research in this area to pursue a more general control theory for networked dynamic systems.

Within the framework of this setup, the use of graph theory as a tool for analysis is recognized as the correct mathematical abstraction to study these systems. Perhaps the most celebrated result in this direction is the relationship of the algebraic connectivity of a graph, sometimes referred to as the Fiedler eigenvalue [7], to the convergence rate of the dynamic system. The importance of this property has been extended in many directions, including convergence analysis for random graphs [8] and graphs with communication delays and switching topologies [9]. Various modifications to the consensus protocol have also led to more general system theoretic notions such as controllability and observability [10,11] and input–output properties [12–14].

While the analysis of consensus systems has matured, work related to the design of consensus protocols and the characterization of system performance beyond that of its convergence rate have not been as deeply studied. As the consensus protocol becomes more integrated into complex real-world networks, the importance of design from an optimality perspective becomes crucial. A fundamental challenge for the design of consensus networks, however, relates to the computational complexity of solving combinatorial problems. A common approach to this problem is to consider optimization over weighted graphs or other convex relaxations [15–17]. However, most results that deal with the design of networks focus on the optimization of the Fiedler eigenvalue, aligning with the mainstream results related to connectivity and convergence rates [18,19].

An important extension of these works, therefore, is the introduction of more general notions of system performance in coordination with the design of these networks. A first step in this direction is to include exogenous inputs in the form of noises and disturbances into the consensus protocol. Such models have been considered in [17,20,21] where noises were introduced in either the process or measurement of the consensus protocol. This more general model of the consensus protocol better reflects their use in real-world systems. Indeed, multi-agent systems relying on

☆ The authors thank the German Research Foundation (DFG) for financial support within the Cluster of Excellence in Simulation Technology and the Priority Programme 1305 "Control Theory of Digitally Networked Dynamical Systems".

* Corresponding author. Tel.: +972 4 829 3196.
E-mail addresses: dzelazo@technion.ac.il (D. Zelazo), simone.schuler@ist.uni-stuttgart.de (S. Schuler), frank.allgower@ist.uni-stuttgart.de (F. Allgöwer).

0167-6911/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysconle.2012.10.014

relative sensing, such as formation flying or sensor fusion, will depend on sensors and actuators that are imperfect.

Noises in the consensus protocol lead to a random walk of the agreement value, and attempts to compensate for this includes the design of edge weights [17], or the introduction of a time-varying gain on the control to effectively reject the noise asymptotically [20]. When the consensus protocol is corrupted by noise, the $\mathcal{H}_2$ system norm provides a measure of how the noises affect the asymptotic deviation of each node's state from a consensus state. This performance metric has been used to study, for example, the leader selection problem for consensus networks [22]. The $\mathcal{H}_2$ performance of large scale networked systems and fractal graphs were considered in [23,24]. Other works have considered this metric for consensus networks over directed graphs [25], spanning trees [26], and biochemical networks [27].

However, many of these results do not exploit the underlying combinatorial properties of the graph as it relates to these metrics. We emphasize a distinction between spectral properties of the graph, i.e. the eigenvalues of the Laplacian matrix, and combinatorial properties of a graph such as path lengths and cycles. Indeed, spectral properties introduce a layer of abstraction to the underlying graph that makes more tangible design issues, such as edge costs and distances, less intuitive.

A thorough treatment in this direction was recently given in [28] via the introduction of the *Edge Laplacian* and its corresponding *edge agreement problem*. The edge Laplacian is a variant of the graph Laplacian that provides a more transparent understanding of how spanning trees and cycles affect certain algebraic properties of a graph. When the consensus protocol is analyzed using this construction, clear graph theoretic interpretations of the $\mathcal{H}_2$ norm of the system can be derived [28].

An unresolved question from [28], however, was the precise role that cycles play in the performance of consensus networks. The main result from [28] showed that the $\mathcal{H}_2$ performance is determined by the inverse of a matrix related to the cycle structure of the graph. However, the precise structure of this matrix was not considered, and this work contributes in that regard. It is well known that the addition of cycles in a graph will increase its algebraic connectivity [29],[1] however, a similar result has not been found for other performance metrics. A fundamental contribution of this work, therefore, is an analytic characterizations of how cycles affect the $\mathcal{H}_2$ performance of consensus networks. In this direction, we provide new interpretations for the role of cycles as related to algebraic properties of the edge Laplacian, and dynamic properties of the consensus protocol. In particular, we show that the $\mathcal{H}_2$ norm of the consensus protocol always improves with the addition of cycles. We provide an exact characterization of how the addition of one or two cycles improves the performance. The improvement is proportional to the inverse of the cycle lengths and also related to the number of edges that are shared between cycles. This establishes a strong connection between *combinatorial properties* of the underlying graph and *dynamic properties* of the corresponding protocol.

The analytic results are then used to formulate a synthesis problem for consensus networks. The problem considers the task of adding a fixed number of edges to an existing consensus network over a spanning tree that leads to the greatest improvement in performance. A first approach to this problem leads to a mixed-integer program, generally considered a hard problem to solve due to its combinatorial nature. This combinatorial problem can be formulated as an ℓ_0-optimization problem. By combining graph theoretic insights with results from compressed sensing [30,31], we reformulate the problem as a reweighted ℓ_1-optimization problem. The ℓ_1-optimization problems are well known to achieve sparse solutions [32–34] and we show that this convex relaxation of the original mixed-integer problem gives very good results. We also highlight how the weighting mechanism used in the ℓ_1 formulation provides an important tuning parameter for design. The analytic results of this work provide intuition for appropriate weighting functions, including cycle length weighting and cycle correlation weighting. This formulation allows to consider additional performance criteria including maximizing the algebraic connectivity of the graph. These results are demonstrated via some simulation examples.

This paper is organized as follows. Section 2 reviews the fundamental properties of the edge Laplacian and provides results relating to algebraic properties of this matrix and cycles in this graph. The edge agreement problem is given in Section 3. These results are then applied in Section 4 to derive the $\mathcal{H}_2$ performance of the agreement protocol as a function of the edges in the graph. The synthesis problem and ℓ_1 formulation is given in Section 5, and numerical simulations are presented in Section 6. Finally, we offer some concluding remarks in Section 7.

1.1. Notations

The notation used is standard. The set of real numbers is denoted by $\mathbb{R}$. For a vector $x \in \mathbb{R}^n$, its transpose is given by x^T and the ith component by x_i; the ijth element of a matrix A is denoted $[A]_{ij}$. The trace of a matrix is denoted $\mathbf{tr}[\cdot]$. The null space and range space of a matrix is denoted as $\mathcal{N}(A)$ and $\mathcal{R}(A)$ respectively. A symmetric matrix is positive definite (semi-definite) if all its eigenvalues are positive (non-negative), and is denoted $A > 0$ ($A \geq 0$); the linear matrix inequality $A > B$ ($A \geq B$) is equivalent to $A - B > 0$ ($A - B \geq 0$). The cardinality of a set M is denoted as $|M|$. The ℓ_1 norm of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_1 = \sum_i |x_i|$. The ℓ_0-norm of a vector is defined as $\|x\|_0 = |\{i \mid x_i \neq 0\}|$, the number of non-zero elements in the vector x. Note that in fact, the ℓ_0-norm is not a true norm, but it is commonly referred to as a norm in the literature and we adopt that convention in this work.

2. Cycles and the edge Laplacian

As discussed in the introduction, graph theory plays a central role in the analysis of consensus networks. In particular, the consensus protocol is described in terms of a certain algebraic representation of the underlying communication graph known as the (graph) Laplacian. The work presented in here, however, relies on an alternative representation that we term the *edge Laplacian* [28]. In this section, we review the construction of the edge Laplacian and focus the presentation on the role cycles play for its algebraic properties.

2.1. Graphs, spanning trees, and cycles

We first provide a brief review of concepts from graph theory [29]. A *graph*, denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consists of a set of *nodes* $\mathcal{V} = \{v_1, \ldots, v_n\}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, describing the incidence relation between pairs of nodes. Unless otherwise stated, we assume that $|\mathcal{V}| = n$. In this work we deal with *undirected* graphs but at times will assign an arbitrary orientation to each edge. In this way, we denote an edge $e \in \mathcal{E}$ with the ordered pair $(v_i, v_j) \in \mathcal{V} \times \mathcal{V}$ as the *directed edge* connecting v_i to v_j; we also use the notation $v_i \sim v_j$ to denote that these two nodes are connected (or adjacent). A *path* is a sequence of distinct nodes such that consecutive nodes are adjacent to each other. If the initial and terminal node of a path are the same, it is called a *cycle*. The length of a path (cycle) is the number of edges traversed in the path

[1] More precisely, the algebraic connectivity of a graph is a non-decreasing function of the number of edges.

sequence. For example, a *triangle* is a cycle of length 3. The *diameter* of a graph, denoted **diam**$[\mathcal{G}]$, is the maximum distance[2] between any two nodes in a graph. A graph is *connected* if there exists a path between any pair of nodes; otherwise it is called *disconnected*. In this work, we always assume connected graphs.

A graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a *subgraph* of $\mathcal{G}$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$; equivalently, we write $\mathcal{G}' \subseteq \mathcal{G}$. A *spanning tree* of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a connected cycle-free subgraph on the same node set, denoted $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau) \subseteq \mathcal{G}$. Consequently, $|\mathcal{E}_\tau| = |\mathcal{V}| - 1$ for any choice of spanning tree. For a given spanning tree $\mathcal{T}$, the set $\mathcal{E}_c = \mathcal{E} \backslash \mathcal{E}_\tau$ contains all the edges in $\mathcal{G}$ that are not in $\mathcal{T}$. These edges must therefore complete the cycles of the graph, and we denote the *cycle subgraph of* $\mathcal{G}$ as $\mathcal{C} = (\mathcal{V}, \mathcal{E}_c) \subseteq \mathcal{G}$. Note that the cycle subgraph depends on the choice of the spanning tree, formalized by the relation $\mathcal{T} \cup \mathcal{C} = \mathcal{G}$. The *complement* of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is denoted $\overline{\mathcal{G}} = (\mathcal{V}, \overline{\mathcal{E}})$ such that $\overline{\mathcal{E}} = \{e \in \mathcal{V} \times \mathcal{V} \,|\, e \notin \mathcal{E}\}$. A particular graph of importance to this work is the *complete graph* on n nodes, denoted K_n. The complete graph contains all possible edges, which for undirected graphs has cardinality $n(n-1)/2$. For any given graph $\mathcal{G}$ on n nodes, the relationship $\mathcal{G} \cup \overline{\mathcal{G}} = K_n$ holds. This relation proves useful when considering the complement of a spanning tree, $\overline{\mathcal{T}}$, as it describes all possible cycles that can be created from that particular spanning tree.

Graphs also admit several useful algebraic representations. The *incidence matrix* of the graph $\mathcal{G}$, $E(\mathcal{G}) \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$, is a $\{0, \pm 1\}$-matrix with rows and columns indexed by the vertices and edges of $\mathcal{G}$ such that $[E(\mathcal{G})]_{ik}$ has the value '+1' if node i is the initial node of edge k, '−1' if it is the terminal node, and '0' otherwise. The (graph) Laplacian is a symmetric and positive semi-definite matrix defined using the incidence matrix as [29]

$$L(\mathcal{G}) = E(\mathcal{G})E(\mathcal{G})^T. \tag{1}$$

Note that while the incidence matrix encodes edge directions, the Laplacian loses such information. In this work, we denote the eigenvalues of the Laplacian as

$$0 = \lambda_1(\mathcal{G}) \leq \lambda_2(\mathcal{G}) \leq \cdots \leq \lambda_{|\mathcal{V}|}(\mathcal{G}).$$

The eigenvector associated with $\lambda_1(\mathcal{G})$ is the all-ones vector, $\mathbb{1} \in \mathbb{R}^{|\mathcal{V}|}$. The algebraic connectivity of the graph $\lambda_2(\mathcal{G})$, sometimes referred to as the Fiedler eigenvalue, is strictly positive if and only if the graph is connected [7,29]. The Laplacian for the complete graph K_n can be expressed in terms of the Laplacian for any spanning tree $\mathcal{T}$ and the corresponding cycles $\mathcal{C} = \overline{\mathcal{T}}$ as

$$L(K_n) = L(\mathcal{T}) + L(\mathcal{C}) = nI - \mathbb{1}\mathbb{1}^T. \tag{2}$$

Using an appropriate labeling of the edges in the graph, we can always express the incidence matrix in terms of the subgraphs $\mathcal{T}$ and $\mathcal{C}$ for a particular choice of spanning tree,

$$E(\mathcal{G}) = \begin{bmatrix} E(\mathcal{T}) & E(\mathcal{C}) \end{bmatrix}.$$

This representation aids in the interpretation of several results relating the sub-graphs $\mathcal{T}$ and $\mathcal{C}$. For example, a *signed path vector* $\xi \in \mathbb{R}^{\mathcal{E}}$ is a $\{0, \pm 1\}$-vector corresponding to a path in $\mathcal{G}$, such that ξ_i takes the value '+1' ('−1') if edge $e_i \in \mathcal{E}$ is traversed positively (negatively), and '0' otherwise. Any path, therefore, can be expressed using only edges from the sub-graph $\mathcal{T}$. Observe that the length of a path can be computed from its signed path vector as $\xi^T\xi$. Furthermore, a cycle can be expressed using exactly one edge from $\mathcal{C}$, and the remaining edges from $\mathcal{T}$. This is formalized by the following result, establishing a strong connection between algebraic properties of the incidence matrix and properties of the underlying graph.

Theorem 1 (*[29]*)**.** *For a connected graph $\mathcal{G}$, the null space of $E(\mathcal{G})$ is spanned by all the linearly independent signed path vectors corresponding to the graph cycles.*

[2] The distance between two nodes is the length of the shortest path connecting them.

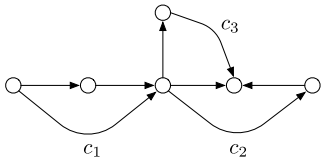


Fig. 1. A graph with cycles c_1, c_2 and c_3. The cycle c_1 is edge-disjoint with c_2 and c_3, while c_2 and c_3 are correlated.

Theorem 1 implies that the incidence matrix corresponding to the cycle subgraph can be expressed as a linear combination of the edges in the spanning tree. Formally, we define the matrix $T_{(\mathcal{T},\mathcal{C})} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}_c|}$ as [28]

$$T_{(\mathcal{T},\mathcal{C})} = \left(E(\mathcal{T})E(\mathcal{T})^T\right)^{-1} E(\mathcal{T})^T E(\mathcal{C}), \tag{3}$$

satisfying

$$E(\mathcal{T})T_{(\mathcal{T},\mathcal{C})} = E(\mathcal{C}).$$

The matrix $T_{(\mathcal{T},\mathcal{C})}$, therefore, encodes information related to the cycles that can be formed from the spanning tree $\mathcal{T}$. To further aid in the following exposition, we express $T_{(\mathcal{T},\mathcal{C})}$ in terms of its columns, $T_{(\mathcal{T},\mathcal{C})} = \begin{bmatrix} c_1 & \cdots & c_{|\mathcal{E}_c|} \end{bmatrix}$, and using a slight abuse in convention, we will also refer to the ith column of $T_{(\mathcal{T},\mathcal{C})}$ as the ith cycle of the graph $\mathcal{G}$. Similarly, we will refer to the ith column of $E(\mathcal{T})$ with τ_i as the ith edge in the spanning tree. This matrix is also referred to as the *Tucker representation* of a graph, used in network optimization communities [35]. A surprising result is that the number of spanning trees that can be found in a graph, $\tau(\mathcal{G})$, can be determined as [29]

$$\tau(\mathcal{G}) = \det[I + T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T]. \tag{4}$$

At times, we will refer to the matrix $R_{(\mathcal{T},\mathcal{C})} = \begin{bmatrix} I & T_{(\mathcal{T},\mathcal{C})} \end{bmatrix}$. Using this notation, note that

$$E(\mathcal{G}) = E(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}.$$

We explore now additional properties of the matrix $T_{(\mathcal{T},\mathcal{C})}$ and its variations. While (3) suggests that a matrix inverse is required to compute $T_{(\mathcal{T},\mathcal{C})}$, this is in fact unnecessary. Indeed, if the spanning tree is given, one only need construct a signed path vector corresponding to the edges in the tree that form a cycle with a corresponding edge in the cycle subgraph; this signed path vector will form a column of the matrix $T_{(\mathcal{T},\mathcal{C})}$. This is discussed in more detail as a consequence of the *basis theorem* in [35].

The previous discussion suggests that $T_{(\mathcal{T},\mathcal{C})}$ also encodes information related to the length of each cycle, denoted $l(c_i)$, and correlations between different cycles in $\mathcal{G}$. In this direction, we first define the notion of *edge-disjoint cycles* and *correlated cycles*.

Definition 1. Two cycles are said to be *edge-disjoint* if they do not have any edges in common.

Definition 2. Two cycles are said to be *correlated* if they are not edge-disjoint.

If two cycles c_i and c_j are correlated, the quantity s_{ij} is the number of edges the cycles share. Fig. 1 shows a graph with three cycles illustrating the definitions.

Proposition 1. *The matrix $T_{(\mathcal{T},\mathcal{C})}^T T_{(\mathcal{T},\mathcal{C})}$ encodes the following information about the cycles in $\mathcal{G}$.*

1. $[T_{(\mathcal{T},\mathcal{C})}^T T_{(\mathcal{T},\mathcal{C})}]_{ii} = c_i^T c_i = l(c_i) - 1$.
2. $[T_{(\mathcal{T},\mathcal{C})}^T T_{(\mathcal{T},\mathcal{C})}]_{ij} = c_i^T c_j = 0$ *if and only if cycles c_i and c_j are edge-disjoint.*
3. $[T_{(\mathcal{T},\mathcal{C})}^T T_{(\mathcal{T},\mathcal{C})}]_{ij} = \pm s_{ij}$ *if and only if cycles c_i and c_j are correlated.*

Proof. The ith column of $T_{(\mathcal{T},\mathcal{C})}$ is a signed path vector with length $c_i^T c_i$. The length of the cycle formed using this signed path vector must be $l(c_i) = c_i^T c_i + 1$. For the next statement, first consider the case where $c_i^T c_j = 0$. This implies that the two vectors are orthogonal which can in general describe two situations. The first possibility is whenever $[c_i]_k = 0$ ($[c_i]_k \in \{0, \pm 1\}$) then $[c_j]_k \in \{0, \pm 1\}$ ($[c_j]_k = 0$) which is precisely when the cycles are edge-disjoint. The other possibility occurs if for each edge e_k with $[c_i]_k[c_j]_k = +1$ ($[c_i]_k[c_j]_k = -1$), there exists another edge e_s such that $[c_i]_s[c_j]_s = -1$ ($[c_i]_s[c_j]_s = +1$); note that this implies that the cycles are correlated and must share an even number of edges. We will now show that this can never be possible. Denote the directed path of length p from nodes v to v' as the sequence of edges $\{\tau_i,\ i = 1, \ldots, p\}$ in the spanning tree subgraph $\mathcal{T}$, with τ_i adjacent to τ_{i+1}. Similarly, denote the directed path of length p' from nodes u to u' as the sequence of edges $\{\tilde{\tau}_i,\ i = 1, \ldots, p'\}$ in the spanning tree subgraph $\mathcal{T}$ with $\tilde{\tau}_i$ adjacent to $\tilde{\tau}_{i+1}$. Furthermore, assume the edges $(v, v'), (u, u') \in \mathcal{C}$; that is they are edges in the cycle sub-graph. Observe that if the cycles are correlated, the shared edges must be a sequence of adjacent edges.[3] Assume without loss of generality that the shared edges are $\{\tau_l, \ldots, \tau_{l+h}\}$ for some odd number h, and this sequence provides a directed path from a node q to a node q' (i.e., the node q is used by edge τ_l, and the node q' is used by edge τ_{l+h}). The path from v to v' must either contain the directed path from q to q', or from q' to q. The same is true for the path from u to u'. This implies that $c_i^T c_j = \pm(h+1)$, leading to a contradiction. For the other direction, assume that two cycles c_i and c_j are edge disjoint. Then the signed path vector corresponding to each cycle have no non-zero elements in common, and $c_i^T c_j = 0$. The proof of the third statement follows from the second, with the sign indicating if the direction through the spanning tree is the same or opposite, and observing that $h + 1$ is s_{ij}. □

Proposition 2. *The matrix $T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T$ encodes the following information about the cycles in $\mathcal{G}$.*

1. *$[T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T]_{ii}$ is the number of times edge τ_i is used to construct the cycles in $\mathcal{C}$.*
2. *$[T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T]_{ij}$ is $\pm$ the number of times edges τ_i and τ_j are used in the same cycle. The sign of the entry is positive if both edges are traversed in the same direction, and negative otherwise.*
3. *$(I + T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T)$ is invertible.*

Proof. The first and second statements follows directly from the structure of $T_{(\mathcal{T},\mathcal{C})}$. The sign condition of the second statement follows from a similar argument used in the proof of Proposition 1. The inevitability of $(I + T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T)$ follows from the fact that $T_{(\mathcal{T},\mathcal{C})}^T T_{(\mathcal{T},\mathcal{C})}$ is a positive-semi definite matrix, and has the same non-zero eigenvalues and rank as $T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T$. □

The properties described in Propositions 1 and 2 will prove useful in the sequel.

2.2. The edge Laplacian

The *edge Laplacian* is a $|\mathcal{E}| \times |\mathcal{E}|$ symmetric matrix defined as [28]

$$L_e(\mathcal{G}) := E(\mathcal{G})^T E(\mathcal{G}). \tag{5}$$

One of the results in [28] showed that the edge Laplacian is related to the graph Laplacian via a similarity transformation. We summarize the results here and refer the reader to [28] for the proof.

[3] This must be true, otherwise there would be two distinct paths connecting the edges that are not in sequence, implying that one of the edges in $\{\tau_i\}$ or $\{\tilde{\tau}_i\}$ must not be in the spanning tree subgraph.

Theorem 2 ([28]). *The graph Laplacian for a connected graph $L(\mathcal{G})$ containing cycles with an arbitrary but fixed spanning tree $\mathcal{T}$ is similar to the matrix*

$$\begin{bmatrix} L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T & 0 \\ 0 & 0 \end{bmatrix},$$

where $L_e(\mathcal{T})$ and $R_{(\mathcal{T},\mathcal{C})} = \begin{bmatrix} I & T_{(\mathcal{T},\mathcal{C})} \end{bmatrix}$ are defined in (5) and (3) respectively.

The edge Laplacian for a graph with cycles, $L_e(\mathcal{G})$, is similar to the matrix

$$\begin{bmatrix} L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T & 0 \\ 0 & \mathbf{0} \end{bmatrix},$$

where the block-matrix of zeros is square with dimension equal to the number of independent cycles in the graph.

The edge Laplacian for any graph, $L_e(\mathcal{G})$, is similar to the bordered graph Laplacian

$$\begin{bmatrix} L(\mathcal{G}) & 0 \\ 0 & \mathbf{0} \end{bmatrix},$$

where the block-matrix of zeros is square with dimension equal to the number of independent cycles in the graph minus one.

We refer to the matrix $L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T \in \mathbb{R}^{|\mathcal{E}_\mathcal{T}|\times|\mathcal{E}_\mathcal{T}|}$ as the *essential edge Laplacian*. A direct consequence of Theorem 2 is that for any connected graph the essential edge Laplacian has only positive eigenvalues, and they are precisely the non-zero eigenvalues of $L(\mathcal{G})$. Furthermore, when the underlying graph is a tree (i.e. $\mathcal{G} = \mathcal{T}$), then $L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T = E(\mathcal{T})^T E(\mathcal{T})$ is a symmetric positive-definite matrix.

Corollary 1. *The essential edge Laplacian for the complete graph K_n is*

$$L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T = nI. \tag{6}$$

Proof. For notational simplicity we denote $E_\tau = E(\mathcal{T})$ and $E_c = E(\mathcal{C})$. Using (2) and (3) obtains the following chain of equalities,

$$\begin{aligned} E_\tau(L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T)E_\tau^T &= E_\tau E_\tau^T E_\tau(I + T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T)E_\tau^T \\ &= L(\mathcal{T})^2 + L(\mathcal{T})L(\mathcal{C}) \\ &= L(\mathcal{T})\left(L(\mathcal{T}) + L(\mathcal{C})\right) \\ &= L(\mathcal{T})\left(nI - \mathbb{1}\mathbb{1}^T\right) = nL(\mathcal{T}). \end{aligned}$$

The last term implies that $L_e(\mathcal{T})R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T = nI$. □

The essential edge Laplacian is the main tool used to derive an edge variant of the consensus protocol. It is important to emphasize that the similarity transformation discussed in Theorem 2 preserves both the algebraic properties of the Laplacian, along with structural properties relating the graph to its matrix representation. The benefit of this transformation is explored in the sequel.

3. The edge agreement problem

The standard consensus model is based on a collection of n single integrator agents that exchange relative state information over a communication graph to generate a control. The model is usually presented as an autonomous system with no noises or disturbances [1],

$$\dot{x}(t) = -L(\mathcal{G})x(t). \tag{7}$$

Here, the vector $x(t) \in \mathbb{R}^n$ is the concatenated state of each agent, and $L(\mathcal{G})$ is the Laplacian matrix.

A two-port interpretation of the consensus protocol provides a framework for considering the presence of exogenous inputs, such as reference signals and noises entering the measurement and

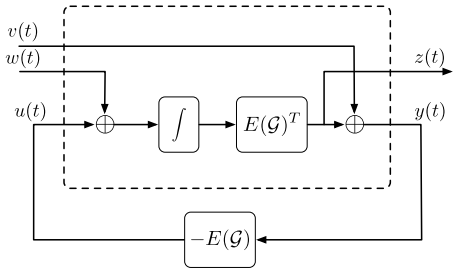


Fig. 2. Consensus as a 2-port feedback configuration.

process.[4] This representation is pictured in Fig. 2, and is described as

$$\begin{cases} \dot{x}(t) = u(t) + w(t) \\ z(t) = E(\mathcal{G})^T x(t) \\ y(t) = E(\mathcal{G})^T x(t) + v(t), \end{cases} \tag{8}$$

with the control defined as

$$u(t) = -E(\mathcal{G}) y(t).$$

The closed-loop system thus has the dynamics

$$\Sigma(\mathcal{G}) : \begin{cases} \dot{x}(t) = -L(\mathcal{G}) x(t) + \begin{bmatrix} I & -E(\mathcal{G}) \end{bmatrix} \begin{bmatrix} w(t) \\ v(t) \end{bmatrix} \\ z(t) = E(\mathcal{G})^T x(t). \end{cases} \tag{9}$$

Remark 1. The two-port representation of the consensus protocol is meant to illustrate the underlying mechanism of the dynamics; it transparently reveals how disturbances can enter into the consensus model and also shows the distributed nature of the dynamics. The model (8) is closely related to the noise-corrupted models given in [17,20,21], with the main difference that we consider simultaneously noises at the measurement and the process.

Observe that this more general model is not a minimal realization of the system. Indeed, the system has an unobservable mode in the direction of the $\mathbb{1}$ vector [28,36]. Furthermore, due to the eigenvalue at the origin of the state matrix, certain system norms are not meaningful (i.e. the $\mathcal{H}_2$ norm of the system (9) is unbounded). As discussed in [28], the results of Theorem 2 can be used to define a state transformation leading to a minimal realization of the system; the state matrix then becomes the essential edge Laplacian.[5]

$$\Sigma_e(\mathcal{G}) : \begin{cases} \dot{x}_\tau(t) = -L_e(\mathcal{T}) R_{(\mathcal{T},\mathcal{C})} R_{(\mathcal{T},\mathcal{C})}^T x_\tau(t) \\ \qquad + \begin{bmatrix} E(\mathcal{T})^T & -L_e(\mathcal{T}) R_{(\mathcal{T},\mathcal{C})} \end{bmatrix} \begin{bmatrix} w(t) \\ v(t) \end{bmatrix} \\ z(t) = x_\tau(t). \end{cases} \tag{10}$$

Here, the transformed state vector $x_\tau(t) \in \mathbb{R}^{|\mathcal{E}_\tau|}$ can be interpreted as a state associated with the edges of the spanning tree $\mathcal{T}$. The system (10) is referred to as the *edge agreement problem*. An added benefit of this system is the state matrix is now Hurwitz; i.e., all the eigenvalues are in the open left-half of the complex plain. We discuss the implications of this in the sequel. In this work we consider $\mathcal{T}$ as a *skeletal system* for the complete consensus network. In this regard, we only observe the states along the tree, $x_\tau(t)$, as the controlled variable in the edge agreement problem.[6]

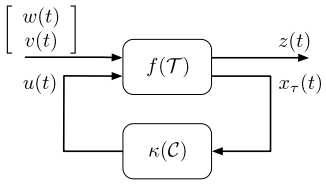


Fig. 3. Cycles as a feedback mechanism.

An interesting interpretation of the minimal system $\Sigma_e(\mathcal{G})$ is that the cycles in the graph can be viewed as an internal feedback mechanism for the system. To elucidate on this idea, consider the following dynamic system over a spanning tree $\mathcal{T}$,

$$f(\mathcal{T}) : \dot{x}_\tau(t) = -L_e(\mathcal{T}) x_\tau + L_e(\mathcal{T}) u(t) + \Gamma \begin{bmatrix} w(t) \\ v(t) \end{bmatrix},$$

with a state-feedback control

$$\kappa(\mathcal{C}) : u(t) = -T_{(\mathcal{T},\mathcal{C})} T_{(\mathcal{T},\mathcal{C})}^T x_\tau(t),$$

and $\Gamma = [E(\mathcal{T})^T \ -L_e(\mathcal{T}) R_{(\mathcal{T},\mathcal{C})}]$. The resulting closed-loop system is the minimal edge-agreement system in (10). When interpreting $T_{(\mathcal{T},\mathcal{C})}$ as an encoder of cycles, we can view the state-feedback term as a generator of cycles; this motivates the above notation for the control law, $\kappa(\mathcal{C})$, as an explicit function of the cycles in the graph. The equivalence of the above system with the edge-agreement system reveals a new interpretation for cycles in consensus networks.

Cycles as a feedback mechanism can lead to a deeper understanding of their role in consensus. Indeed, when presented in a feedback configuration as in Fig. 3, one can conclude that cycles can be used to reduce the sensitivity of the system to external disturbances. This provides a powerful framework for considering problems related to the design of cycles to that of *optimal controller design* [37]. Therefore, based on this new interpretation of cycles as feedback, one can now attempt to cast problems related to the *optimal design of graphs* to that of the *optimal synthesis of feedback controllers*. This sets the stage for presenting a canonical problem for the design consensus networks.

Problem 1 (*Consensus Design*)**.** Consider a consensus network over a spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$ and a set of candidate edges in $\overline{\mathcal{E}}_\tau$. Add k edges to $\mathcal{T}$ from the set $\overline{\mathcal{E}}_\tau$ that leads to the largest improvement in the performance of the edge agreement problem. That is, solve the following optimization problem:

$$\min_{T_{(\mathcal{T},\mathcal{C})} \in \mathbb{R}^{|\mathcal{V}| \times k}} \quad \| \Sigma_e(\mathcal{G}) \|_p, \tag{11}$$

for some p (i.e. $p = 2, \infty$).

Note that this problem is in fact an integer program (the matrix $T_{(\mathcal{T},\mathcal{C})}$ takes values in $\{0, \pm 1\}$). In the sequel, we first characterize how cycles impact the performance of the edge agreement problem, and then present a solution method for solving Problem 1 using an l_1-relaxation method.

[4] In this work we assume the inputs are white Gaussian noises with unit covariance.

[5] Thus, (10) represents the dynamics for only the controllable and observable modes of the system.

[6] As opposed to considering $R_{(\mathcal{T},\mathcal{C})}^T x(t)$ as the controlled variable.

4. Performance of cycles

In this work, we consider the $\mathcal{H}_2$ performance of the edge agreement problem (10), and in particular focus on how the addition of cycles impacts this measure. Analyzing the consensus protocol with this metric is meaningful in the sense that it provides a measure of how noises can affect the asymptotic deviation of each agent from a consensus configuration; i.e., it studies the effect of noises on the *relative states* in the consensus protocol. As discussed in [17,20], noises can lead to a random walk of the consensus value. Indeed, it is a straightforward exercise to verify that the covariance of the average of the system states, $(1/n)\mathbb{1}^T x(t)$, is a linear function of time (and therefore unbounded as $t \to \infty$). Therefore, when considering a noise-corrupted consensus protocol, we are more concerned with how the noises affect deviations from a consensus state, rather than convergence to the *average consensus value*. The analysis given here will provide the analytic foundation for considering the design problem proposed in Problem 1. The $\mathcal{H}_2$ performance was originally considered in [28] and we briefly review its derivation here.

Recall that the $\mathcal{H}_2$ performance of a linear system $\Sigma = (A, B, C)$ can be computed as $\|\Sigma\|_2^2 = \mathbf{tr}[CXC^T]$, where X is the solution of the Lyapunov equation [37],

$$\mathcal{L}(X) = AX + XA^T + BB^T = 0. \tag{12}$$

For the edge agreement problem, the solution of the Lyapunov equation can be written by inspection as [28]

$$X(\mathcal{G}) = \frac{1}{2}\left((R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T)^{-1} + L_e(\mathcal{T})\right). \tag{13}$$

We emphasize that the solution is a function of the underlying graph $\mathcal{G}$, and in particular on the choice of spanning tree and cycles. This can be used to characterize the $\mathcal{H}_2$ performance of the edge agreement problem.

Theorem 3 (*[28]*). *The $\mathcal{H}_2$ performance of edge agreement problem (10) is*

$$\|\Sigma_e(\mathcal{G})\|_2^2 = \mathbf{tr}[X(\mathcal{G})] = \frac{1}{2}\mathbf{tr}\left[(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T)^{-1}\right] + (n-1). \tag{14}$$

A direct corollary of Theorem 3 provides an upper and lower bound for the performance of the edge agreement problem. In particular, we observe that the performance is upper bounded by any choice of spanning tree, and is lower bounded by the complete graph, K_n.

Corollary 2 (*[28]*). *The $\mathcal{H}_2$ performance of the edge agreement problem for an arbitrary connected graph $\mathcal{G}$ is bounded from above and below as*

$$\|\Sigma_e(K_n)\|_2^2 = \frac{n^2-1}{n} \leq \|\Sigma_e(\mathcal{G})\|_2^2 \leq \|\Sigma_e(\mathcal{T})\|_2^2 = \frac{3}{2}(n-1), \tag{15}$$

where $\mathcal{T} \subseteq \mathcal{G}$ is any spanning tree in the graph.

Corollary 2 states that the complete graph K_n will yield the best $\mathcal{H}_2$ performance for the edge agreement problem, and that all trees yield the same upper-bound. This result also indicates that cycles generally improve performance. An open problem, therefore, is to precisely characterize how cycles impact the performance. In this direction, we first present a result showing that the $\mathcal{H}_2$ performance always improves with the addition of cycles.

Theorem 4. *Consider the edge agreement problem where the underlying graph is the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$. Then the $\mathcal{H}_2$ performance of the edge agreement problem on the graph $\mathcal{G} \cup e$ with $e \in \overline{\mathcal{E}}$ is given as*

$$\|\Sigma_e(\mathcal{G}\cup e)\|_2^2 = \|\Sigma_e(\mathcal{G})\|_2^2 - \frac{\mathbf{tr}\left[\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1} cc^T \left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1}\right]}{2\left(1 + c^T\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1}c\right)}, \tag{16}$$

where c is the cycle formed by the new edge e and the tree $\mathcal{T}$.

Proof. Consider any graph $\mathcal{G} \neq K_n$ with spanning tree $\mathcal{T}$. Then the performance of the system $\Sigma_e(\mathcal{G})$ is given by (14). Adding a single edge e creates a new cycle c, and the performance of the augmented system $\Sigma_e(\mathcal{G} \cup e)$ is

$$\|\Sigma_e(\mathcal{G}\cup e)\|_2^2 = \frac{1}{2}\mathbf{tr}\left[\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T + cc^T\right)^{-1}\right] + n - 1. \tag{17}$$

The matrix cc^T is a rank-one matrix, and the matrix inversion in (17) can be computed using the Sherman–Morrison formula [38] for the inverse of a rank-one update as

$$\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T + cc^T\right)^{-1} = \left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1} - \frac{\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1} cc^T \left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1}}{1 + c^T\left(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T\right)^{-1}c}.$$

The second term in the above expression can never be zero and is positive semi-definite. Therefore, the addition of a cycle will always improve the $\mathcal{H}_2$ performance. □

This result can easily be extended to consider the improvement of the performance when adding a subset of edges from a given set of candidate edges.

Corollary 3. *Consider the edge agreement problem with the underlying graph is the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$. Let $M \subseteq \overline{\mathcal{E}}$ be a set of candidate edges, and let $N \subset M$. Then*

$$\|\Sigma_e(\mathcal{G}\cup M)\|_2^2 < \|\Sigma_e(\mathcal{G}\cup N)\|_2^2. \tag{18}$$

Proof. The proof is a direct consequence of Theorem 4 and successive application of the Sherman–Morrison formula for rank-one updates [38]. □

Having established that the addition of cycles always improve the system performance, we now consider combinatorial interpretations, such as cycle lengths, of the previous results.

Theorem 5. *Consider the edge agreement problem where the underlying graph is a spanning tree, $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$, and consider a single edge $e \in \overline{\mathcal{E}}_\tau$. Then the graph $\mathcal{T} \cup e$ has one cycle, c, and the performance of the edge agreement problem is given as*

$$\|\Sigma_e(\mathcal{T}\cup e)\|_2^2 = \|\Sigma_e(\mathcal{T})\|_2^2 - \frac{1}{2}(1 - l(c)^{-1}), \tag{19}$$

where $l(c)$ is the length of the cycle.

Proof. From (3), the matrix $T_{(\mathcal{T},\mathcal{C})}$ can be computed and will have only one column, therefore $T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})}^T$ is a rank-one matrix. To compute the performance of $\Sigma_e(\mathcal{T} \cup e)$, the result of Theorem 3 must be applied. In this setting, $(R_{(\mathcal{T},\mathcal{C})}R_{(\mathcal{T},\mathcal{C})}^T)^{-1}$ can be seen as

the inverse of the identity with a rank-one update. Applying the Sherman–Morrison formula [38] and Proposition 2 leads to

$$(R_{(\mathcal{T},\mathcal{C})}R^T_{(\mathcal{T},\mathcal{C})})^{-1} = \left(I + T_{(\mathcal{T},\mathcal{C})}T^T_{(\mathcal{T},\mathcal{C})}\right)^{-1} = I - \frac{1}{l(c)}T_{(\mathcal{T},\mathcal{C})}T^T_{(\mathcal{T},\mathcal{C})}.$$

Computing the trace yields $\mathbf{tr}[(R_{(\mathcal{T},\mathcal{C})}R^T_{(\mathcal{T},\mathcal{C})})^{-1}] = n - 2 + l(c)^{-1}$. Therefore,

$$\|\Sigma_e(\mathcal{T}\cup e)\|_2^2 = \frac{1}{2}(n-2+l(c)^{-1}) + n - 1 = \frac{3}{2}(n-1) + \frac{1}{2}(l(c)^{-1}-1),$$

and the result follows. □

Theorem 5 immediately leads to the following corollary, bounding the performance increase achievable by adding only one edge to a spanning tree.

Corollary 4. *Consider the edge agreement problem where the underlying graph is a spanning tree, $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$, and consider an edge $e \in \overline{\mathcal{E}}_\tau$. The maximum performance gain is achieved by adding a cycle of length equal to* $\mathbf{diam}[\mathcal{G}]+1$*, while the minimum performance gain is obtained by adding a cycle of length three (i.e., a triangle).*

This result clearly states that longer cycles are better than shorter ones when adding a single edge. Furthermore, it establishes a very strong connection between a purely combinatorial property of the graph, i.e., the length of the cycle, to a system theoretic property of the edge agreement problem, i.e., its $\mathcal{H}_2$ performance. Theorem 5 can be extended to adding multiple edges by using the results of Proposition 1. In particular, the next result describes the change in performance after the addition of k edge-disjoint cycles.

Corollary 5. *Consider the edge agreement problem where the underlying graph is a spanning tree, $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$, and consider adding k edges $e_i \in \overline{\mathcal{E}}_\tau$ for $i = 1, \ldots, k$, such that the new edges add k edge-disjoint cycles, c_i, to the graph. Then the performance of the edge agreement problem is given as*

$$\|\Sigma_e(\mathcal{T}\cup_{i=1}^k e_i)\|_2^2 = \|\Sigma_e(\mathcal{T})\|_2^2 - \frac{1}{2}\left(k - \sum_{i=1}^k (l(c_i))^{-1}\right). \tag{20}$$

Corollary 5 can be used as a guideline for adding edges to a tree with maximum benefit to the $\mathcal{H}_2$ performance. This problem, adding k-edge disjoint cycles to a graph, turns out to be a very difficult combinatorial problem to solve and is related to the *cycle packing problem*, which for general graphs is $\mathcal{NP}$-hard [39].

In many cases, it will not be possible to add only disjoint cycles. Indeed, even adding three cycles to the path graph on 5 nodes will require at least two correlated cycles. In this direction, we first discuss how the addition of two arbitrary cycles affects the performance.

Corollary 6. *Consider the edge agreement problem where the underlying graph is a spanning tree, $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$, and consider adding 2 edges $e_1, e_2 \in \overline{\mathcal{E}}_\tau$, such that the new edges add 2 new cycles. Then the performance of the edge agreement problem is given as*

$$\|\Sigma_e(\mathcal{T}\cup\{e_1,e_2\})\|_2^2 = \|\Sigma_e(\mathcal{T})\|_2^2 - \left(1 - \frac{l(c_1)+l(c_2)}{2(l(c_1)l(c_2)-s_{12}^2)}\right). \tag{21}$$

Proof. The matrix $T_{(\mathcal{T},\mathcal{C})}$ must have two columns and consequently $T_{(\mathcal{T},\mathcal{C})}T^T_{(\mathcal{T},\mathcal{C})}$ has rank 2. Furthermore, the non-zero eigenvalues of $T_{(\mathcal{T},\mathcal{C})}T^T_{(\mathcal{T},\mathcal{C})}$, denoted μ_i, are the same as the non-zero eigenvalues of

$$T^T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})} = \begin{bmatrix} l(c_1)-1 & \pm s_{12} \\ \pm s_{12} & l(c_2)-1 \end{bmatrix}.$$

These eigenvalues can be determined as

$$\mu_i = \frac{1}{2}\left(l(c_1)+l(c_2) \pm \sqrt{l(c_1)^2 + l(c_2)^2 - 2l(c_1)l(c_2) + 4s_{12}^2}\right) - 1.$$

Therefore,

$$\mathbf{tr}\left[(I + T_{(\mathcal{T},\mathcal{C})}T_{(\mathcal{T},\mathcal{C})})^{-1}\right] = n - 3 + \frac{1}{1+\mu_1} + \frac{1}{1+\mu_2} = n - 3 + \frac{l(c_1)+l(c_2)}{l(c_1)l(c_2)-s_{12}^2}.$$

The proof now follows from Theorem 3. □

Note that when the edges are disjoint (i.e., $s_{12} = 0$), Corollary 6 agrees with the results in Corollary 5.

In the context of Corollary 3, one might consider the combinatorial effects of adding multiple correlated cycles. Indeed, through successive application of the rank-one update, the results of Corollary 6 could be extended to an arbitrary number of cycles; this would lead to an expression in the form of a continued fraction. However, already in the simpler case of adding only two cycles, we can conclude that it is advantageous to add long cycles that are minimally correlated. Regardless of how many cycles are added, the above results should also indicate that it may not be trivial to add a fixed number of cycles with the largest impact on the performance. We address this in the sequel through the introduction of an optimization problem aimed at solving (11).

5. Design of cycles

The results of Section 4 provide a clear analytic picture of how cycles impact the $\mathcal{H}_2$ performance of the edge agreement problem. A trivial conclusion that one may arrive at is to always use the complete graph. The complete graph also, for example, has the largest algebraic connectivity ($\lambda_2(\mathcal{G})$) and thus is desirable for other performance indicators. However, as is common in real-world engineering applications, it may not be feasible to implement a consensus network with all possible communication links. A more realistic view of consensus networks involves a constraint limiting the number of edges in the underlying graph. These constraints may derive from both computation, communication, and budgetary restrictions of the designed system.

In this venue, we consider the problem outlined in Problem 1 and propose a convex relaxation of the problem that leads to a semi-definite program. Using this formulation, we then show how additional criteria, such as maximization of the algebraic connectivity, can also be embedded into the problem.

5.1. Cycle design via ℓ_1-optimization

The feedback interpretation of cycles in consensus networks presented in Section 3 represents a powerful paradigm when considering problems in design. The primary difficulty, however, is that the decision to add a new cycle to the tree is a *binary* one; either a candidate edge is added or not added. This means

that Problem 1 falls under the realm of *mixed integer programming* (MIP) [40].

A common approach for solving MIPs is by considering a relaxation of the problem that leads to a convex formulation. For this work, we consider an ℓ_1-relaxation of the problem. Before discussing the relaxation, we first formulate a more detailed description of the problem (11).

To begin, note that given a spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\tau)$, all the candidate edges that can be added belong to the set $\overline{\mathcal{E}}_\tau$. In particular, this set will contain exactly $|\overline{\mathcal{E}}_\tau| = (1/2)(n-1)(n-2)$ edges. Furthermore, note that any graph $\mathcal{G} \subset K_n$ can be obtained by the deletion of edges from the complete graph K_n. By assigning a weight $w_i \in \{0, 1\}$ to each edge in $\overline{\mathcal{E}}_\tau$ we are able to represent any graph with spanning tree $\mathcal{T}$ via an appropriate choice of weights.

Let $T_{(\mathcal{T},\overline{\mathcal{T}})}$ be defined as in (3), describing all cycles that can be created from the spanning tree $\mathcal{T}$. Then the essential edge Laplacian for any graph $\mathcal{G}$ such that $\mathcal{T} \subseteq \mathcal{G}$ can expressed as

$$L_e(\mathcal{T})\left(I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})}\right), \tag{22}$$

where $W = \mathbf{diag}\{w_1, \ldots, w_{|\overline{\mathcal{E}}_\tau|}\}$, and $w_i = 1$ only for edges in $\mathcal{G}$.

Recall now from Theorem 3 that the performance of the edge agreement problem relates to the trace of the inverse of $\left(I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})}\right)$. Therefore, Problem 1 can now be seen as the MIP

$$\min_{w_i} \ \mathbf{tr}\left[\left(I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})}\right)^{-1}\right] \tag{23a}$$

$$\text{s.t.} \ \sum_i w_i = k \tag{23b}$$

$$w_i \in \{0, 1\}. \tag{23c}$$

The constraint (23b) is used to specify how many edges are allowed to be used for cycle design.

The objective (23a) is a non-linear function of the variables w_i. However, by introducing a new variable, the minimization problem can be converted to the mixed-integer semi-definite program. Consider the symmetric matrix $M \in \mathbb{R}^{|\mathcal{E}_\tau| \times |\mathcal{E}_\tau|}$ with $M \geq 0$, then (23) can be written as

$$\min_{M, w_i} \ \mathbf{tr}[M] \tag{24a}$$

$$\text{s.t.} \ \begin{bmatrix} M & I \\ I & I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T_{(\mathcal{T},\overline{\mathcal{T}})} \end{bmatrix} \geq 0 \tag{24b}$$

$$\sum_i w_i = k \tag{24c}$$

$$w_i \in \{0, 1\}. \tag{24d}$$

This can be seen by noting that

$$\mathbf{tr}\left[\left(I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})}\right)^{-1}\right] \leq \mu$$

is equivalent to the matrix inequality

$$\left(I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})}\right)^{-1} \leq M, \tag{25}$$

where M is some positive semi-definite matrix satisfying $\mathbf{tr}[M] \leq \mu$; this results from the fact that the trace operator is monotonic under matrix inequalities [37]. Applying the Schur complement then yields the LMI in (24b). The objective then becomes the minimization of the trace of M, leading to the formulation in (24a).

We are now prepared to discuss a relaxation for this problem. The most common approach for relaxation of $\{0, 1\}$-optimization problems is to relax the constraints on the weights and allow them to take values continuously on the interval $[0, 1]$. This relaxation results in a convex formulation, however there is no guarantee that the solution will be integer, or for that matter, even sparse. This has several ramifications. On the one hand, solutions will lead to a *weighted* agreement problem. More importantly, the constraint (23b) now takes a different interpretation. To guarantee sparse solutions, we reformulate the objective function (24a) in the following way as

$$\min_{M, w_i} \ \mathbf{tr}[M] + \|w\|_0. \tag{26}$$

Recall that the ℓ_0-norm of the vector is a measure of its sparsity (defined in Section 1.1). In this way, minimizing $\|w\|_0$ attempts to maximize the number of zero-elements in the vector.

This is a common sense approach which simply seeks the sparsest w satisfying the constraints. However, such an approach is of little practical use, since the optimization problem is non-convex and NP-hard as its solution requires a combinatorial search which grows faster than polynomial as the number of candidate edges grow [30]. Similar to the convex relaxation for rank minimization in [41], we will use the convex envelope of $\|w\|_0$ defined next.

Let the map f be defined as $f : \mathbb{X} \mapsto \mathbb{R}$, where $\mathbb{X} \subseteq \mathbb{R}^n$. The convex envelope of f (on $\mathbb{X}$), denoted f_{env}, is defined as the pointwise largest convex function g such that $g(x) \leq f(x)$ for all $x \in \mathbb{X}$.

Lemma 1 (*[42]*). *The convex envelope of the function* $f = \|x\|_0 = \sum_{i=1}^n |\text{sign}(x_i)|$ *on* $\mathbb{X} = \{x \in \mathbb{R}^n \,|\, \|x\|_\infty \leq 1\}$ *is* $f_{\text{env}}(x) = \|x\|_1 = \sum_{i=1}^n |x_i|$.

With this, we can relax the non-convex ℓ_0-minimization in (26) by the convex ℓ_1-minimization

$$\min_{M, w_i} \ \mathbf{tr}[M] + \|w\|_1; \tag{27}$$

note that this can be solved using linear programming. Additionally, this is the best possible convex relaxation since the ℓ_1-norm is the convex envelope of the ℓ_0-norm.

As described in [32], *reweighted* ℓ_1-minimization can be used to improve the results of the minimization. In this direction, weights $m_i > 0$ can be assigned to each variable w_i as

$$\min_{M, w_i} \ \mathbf{tr}[M] + \sum_{i=1}^n m_i w_i. \tag{28}$$

For the described design problem, the weights are free parameters. They counteract the influence of the signal magnitude on the ℓ_1-penalty function. If $m_i = 1$ for all i, the weighted ℓ_1-norm reduces to the regular ℓ_1-norm. If the weights m_i are chosen to be inversely proportional to the magnitude of w_i

$$\begin{cases} m_i = 1/|w_i|, & w_i \neq 0 \\ m_i = \infty, & w_i = 0, \end{cases} \tag{29}$$

then the weighted ℓ_1-norm and the ℓ_0-norm coincide. Additionally, in the context of Problem 1, the results of Section 4 indicate that a proper choice of weights can be used to force the solution towards certain graphs. Assigning a large weight to specific edges has the interpretation that those edges are not desirable.

This brings us to the complete ℓ_1-relaxation of the MIP described in (23),

$$\min_{M, w_i} \ \alpha \mathbf{tr}[M] + (1 - \alpha) \sum_i m_i w_i \tag{30a}$$

$$\text{s.t.} \ \begin{bmatrix} M & I \\ I & I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})} \end{bmatrix} \geq 0 \tag{30b}$$

$$\sum_i w_i = k \tag{30c}$$

$$0 \leq w_i \leq 1. \tag{30d}$$

Here we have also introduced a weighting factor $\alpha \in [0, 1]$ as a tuning parameter for the relative emphasis on each term in the objective function.

We now discuss possible weighting options for the relaxed version. In fact, we emphasize that the choice of weights is an essential step that must be determined by the designer. In the following, we explore a variety of edge weights based on the results of Section 4. The results of Corollaries 5 and 6 suggest that long cycles with low correlation are desirable. A first attempt at a weighting function, therefore, is to consider the inverse of the length each cycle will create; that is, the weights should be defined as

$$\text{Long Cycle-length weighting: } m_i^{lc} = \mathbf{diam}[\mathcal{G}] + 1 - l(c_i) \tag{31}$$

where c_i is the ith column of the matrix $T_{(\mathcal{T},\mathcal{C})}$. This will place a greater emphasis on adding longer cycles in the graph. Note, however, that this would not prevent the optimization from selecting long cycles that are highly correlated.

On the other hand, it may also be desirable to encourage cycles that are short. This might arise, for example, in scenarios where sensing or communication across long distances is too costly. In this direction, we can consider weights of the form

$$\text{Short Cycle-length weighting: } m_i^{sc} = l(c_i). \tag{32}$$

Another weighting option is to focus on the expected number of correlated edges each cycle contains. This value can be determined using the results of Proposition 1.

$$\text{Cycle-correlation weighting: } m_i^{s} = \frac{1}{|\mathcal{E}_c|} \sum_{j \neq i} \left| \left[T_{(\mathcal{T},\mathcal{C})} \right]_{ij} \right|. \tag{33}$$

Here, edges that are likely to be highly correlated with many other edges will receive a larger weight. Observe that the weights m_i^{lc} and m_i^{s} are therefore inversely proportional; the longer a cycle is the more likely it is to be highly correlated with other cycles. This can give insight into a third possible weighting that attempts to balance both the cycle length and correlations.

$$\text{Cycle-length/correlation weighting: } m_i(s) = \beta m_i^{lc} m_i^{s}; \tag{34}$$

the tuning parameter β can be used as a normalization factor for the weights. A similar weight can be defined for short cycles.

While in practice these weighting functions will lead to sparse solutions for the cycle weights w_i, there may arise situations where the optimization problem does not return an integer solution. In this case, the optimization problem can be iterated until a desired integer solution is obtained. We therefore embed the problem (30) into the following algorithm, originally proposed by [32]:

Algorithm 1 Reweighted ℓ_1-minimization

1. Set $h = 0$ and $m_i^{(0)}$ for $i = 1, \ldots, |\mathcal{E}_c|$ according to the rules (31), (33), or (34).
2. Solve the minimization problem (30) to find the optimal solution $w_i^{(h)}$.
3. Update the weights
$$m_i^{(h+1)} = \frac{1}{w_i^{(h)} + \epsilon},$$
where $\epsilon > 0$ ensures that the inverse is always well defined.
4. Terminate if the solution is integer, otherwise set $h = h + 1$.

Remark 2. In the author's experience, Algorithm 1 reaches an integer solution in 2–3 iterations.

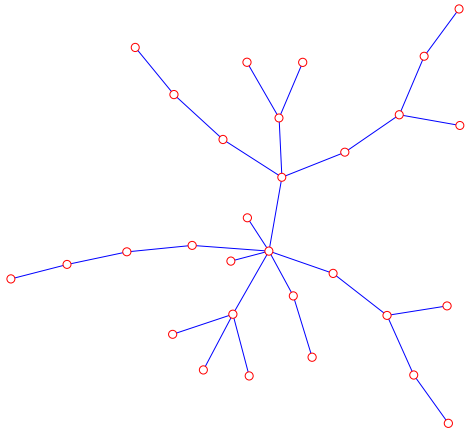


Fig. 4. A spanning tree on 30 nodes.

Remark 3. While the ℓ_1-relaxation leads to a semi-definite program that can be solved in polynomial time, there are still limitations to the solvable problem sizes. In practice, this algorithm performs well for graphs with $|\mathcal{V}| < 100$ nodes (i.e., $|\overline{\mathcal{E}}_\tau| = \mathcal{O}(1000)$). This is a consequence of considering all possible edges in the problem formulation. An area of future work is the development of more efficient algorithms for larger scale graphs using, for examples, algorithms related to cycle packing [43].

As discussed earlier, the first step of the algorithm, i.e., the initial choice of edge weights, greatly influences the solution. From an engineering design stand-point, this can be seen as a favorable feature. Indeed, there may be additional features a designer might want to promote when solving Problem 1 without including additional constraints. This can be accomplished via an appropriate choice of edge weights. This point will be illustrated in the simulations provided in the sequel.

5.2. Cycle design and connectivity maximization

Another advantage of the formulation presented in 5.1 is the ability to embed additional constraints or performance criteria into the problem. One of the most studied performance criteria for consensus networks is the rate of convergence of the system. It is well known that this value is dictated by the algebraic connectivity of the graph, $\lambda_2(\mathcal{G})$ [1].

The algebraic connectivity of the graph can be determined by solving a semi-definite program [15],

$$\max_{\mu} \ \mu \tag{35a}$$

$$\text{s.t. } P^T L(\mathcal{G}) P \geq \mu I, \tag{35b}$$

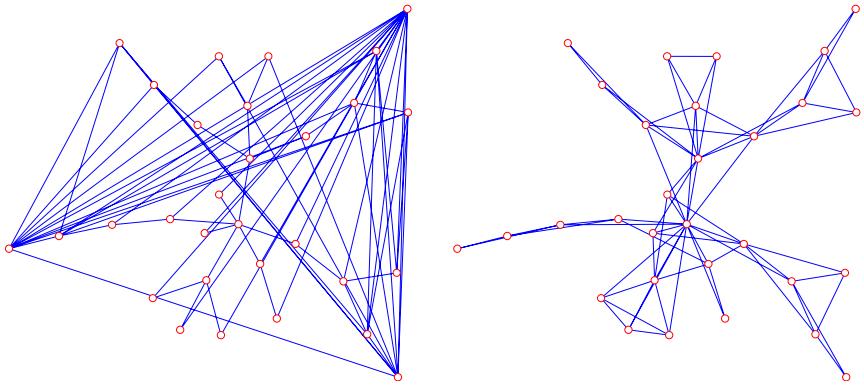
where $L(\mathcal{G})$ is the graph Laplacian, and $P = \text{Im}[\mathbb{1}^\perp]$, the matrix representation of the orthogonal sub-space to the all-ones vector. Note that for a connected graph, $P^T L(\mathcal{G}) P$ is the same size as the essential edge Laplacian.

Similar to the essential edge Laplacian, the graph Laplacian of any graph can be expressed using the complete graph with $\{0, 1\}$ weights on each possible edge. Given a spanning tree, the Laplacian can be expressed as

$$L(\mathcal{G}) = L(\mathcal{T}) + E(\mathcal{T}) T_{(\mathcal{T},\overline{\mathcal{T}})} W T_{(\mathcal{T},\overline{\mathcal{T}})}^T E(\mathcal{T})^T.$$

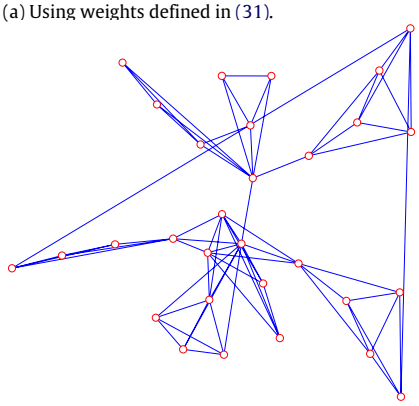
Note that the above equation is linear in the weights. Therefore, this can be embedded into the program (30), leading to the following SDP,

$$\min_{M,\kappa,w_i} \ \alpha_1 \mathbf{tr}[M] - \alpha_2 \kappa + \alpha_3 \sum_i m_i w_i \tag{36a}$$

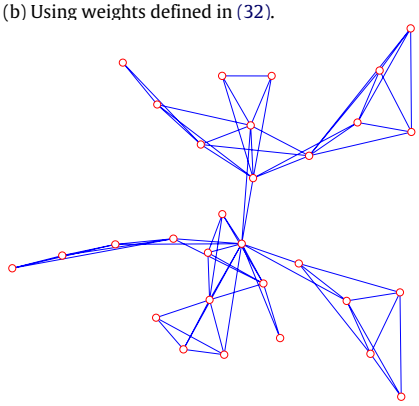


(a) Using weights defined in (31).

(b) Using weights defined in (32).

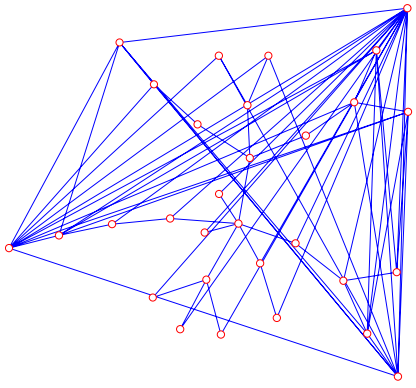


(c) Using weights defined in (34).

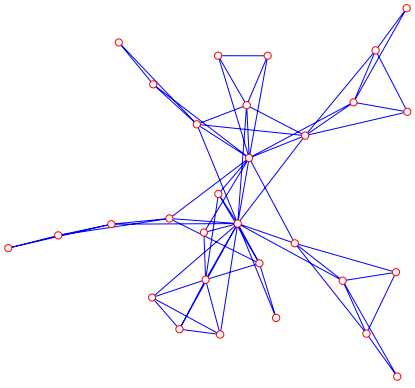


(d) Using identical weights on each edge.

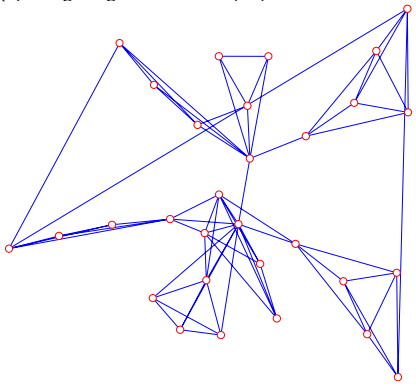
Fig. 5. The weighting function for cycle design has a large affect on the resulting graph.



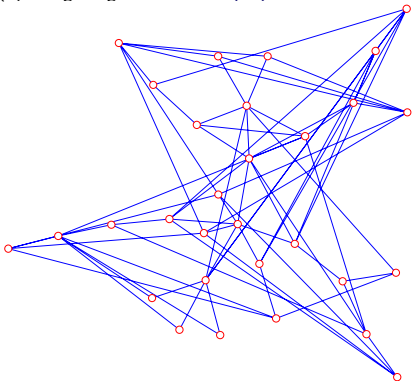
(a) Using weights defined in (31).



(b) Using weights defined in (32).



(c) Using weights defined in (34).



(d) Using identical weights on each edge.

Fig. 6. The weighting function for cycle design has a large affect on the resulting graph.

$$\text{s.t.} \begin{bmatrix} M & I \\ I & I + T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})} \end{bmatrix} \geq 0 \tag{36b}$$

$$\kappa I - P^T E(\mathcal{T}) T_{(\mathcal{T},\overline{\mathcal{T}})} W T^T_{(\mathcal{T},\overline{\mathcal{T}})} E(\mathcal{T})^T P \leq P L(\mathcal{T}) P^T \tag{36c}$$

$$\sum_i w_i = k \tag{36d}$$

$$0 \leq w_i \leq 1. \tag{36e}$$

Remark 4. At the cost of additional conservativeness, the objective function can be reduced to $\alpha_1 \mathbf{tr}[M] + \alpha_2 \sum_i m_i w_i$ and considering κ as a pre-defined constant. This may improve the numerical performance of the optimization.

The program (36) can then be used in Algorithm 1. The resulting solution will be an attempted balance between the rate of convergence of the system and its $\mathcal{H}_2$ performance.

6. Simulation example

In this section we demonstrate the design procedure described in Section 5 with a few numerical examples. For each example we will work with the same spanning tree graph on $|\mathcal{V}| = 30$ nodes, generated randomly in MATLAB; see Fig. 4. The $\mathcal{H}_2$ performance and algebraic connectivity for this graph can be determined as $\|\Sigma(\mathcal{T})\|_2^2 = 43.5$ and $\lambda_2(\mathcal{G}) = 0.07$ and for the complete graph $\|\Sigma(K_n)\|_2^2 = 29.97$ and $\lambda_2(K_n) = 30$. The longest cycle in this graph is determined by its diameter, $\mathbf{diam}[\mathcal{G}] = 10$.

For this example, there are $|\overline{\mathcal{E}}_\tau| = 406$ possible edges that can be added. We will consider Problem 1 while attempting to add 40 new edges. To emphasize the combinatorial difficulty of solving this problem exhaustively, note that there are $\binom{406}{40} \approx 3.6862 \times 10^{55}$ possibilities! First, we compare the performance of Algorithm 1 with the different weighting functions using solvers provided by YALMIP and SeDuMi [44,45]. For m^{lc} and $m^{sc}c$, the weights were normalized to have unit norm. The combined weights were defined as in (34). To further illustrate the effects of the weighting function, we also consider identical weights on each edge, representing a "pure" ℓ_1 objective. The resulting graphs are shown in Fig. 5. Notice that the graph generated by the cycle length weights indeed produces a graph with longer cycles, while the other weighting options tend to favor shorter edge-disjoint cycles. For identical weights on each edge the resulting graph seems, at least qualitatively, to be in between the cycle-length and cycle-correlation results. The resulting performance for each case is given in Table 1.

We now consider the design problem outlined in Section 5.2. The resulting graphs are shown in Fig. 6. It is interesting to observe that including the connectivity constraint tends to favor longer cycles, as demonstrated by comparing the graphs in Figs. 5 and 6. This reveals a subtle result relating, at least through numerical simulation, a correlation between the algebraic connectivity and the length of the cycles. It is also interesting to observe how the addition of the connectivity performance hardly affects the attainable $\mathcal{H}_2$ performance. Note that in all weighting options the value of λ_2 was increased, most significantly for the uniform weights. The resulting performance and value of $\lambda_2(\mathcal{G})$ are also summarized in Table 1.

The simulation examples illustrate both the effectiveness of the design procedure, and the importance of the edge-weight selection. Indeed, as might be expected, the $\mathcal{H}_2$ performance of the system is related to the system eigenvalues, and in particular, $\lambda_2(\mathcal{G})$. While the optimization formulation allows for the simultaneous design of both objectives, it becomes more difficult to understand precisely how the trade-off is emphasized during the numeric evaluation of the algorithm. This formulation, however, does lead to *qualitative* intuitions, as evident by merely glimpsing at the resulting graphs. Furthermore, while the variation of the performance using different edge weights are not dramatic, the resulting graphs to yield significantly different structures. This can be considered an advantage when additional constraints on cycle lengths must be considered. When combined with the analytic results this can be considered a powerful design framework.

Table 1
Summary of $\mathcal{H}_2$ performance and $\lambda_2(\mathcal{G})$ for each simulation example.

ℓ_1 weights	$\mathcal{H}_2$	$\mathcal{H}_2 + \lambda_2(\mathcal{G})$
m_i^{lc}	$\|\Sigma_e(\mathcal{G})\|_2^2 = 36.10$ $\lambda_2(\mathcal{G}) = 0.65$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 36.26$ $\lambda_2(\mathcal{G}) = 0.67$
m_i^{sc}	$\|\Sigma_e(\mathcal{G})\|_2^2 = 34.74$ $\lambda_2(\mathcal{G}) = 0.40$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 34.76$ $\lambda_2(\mathcal{G}) = 0.45$
$m_i = m_i^{lc} m_i^s$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 35.09$ $\lambda_2(\mathcal{G}) = 0.23$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 35.02$ $\lambda_2(\mathcal{G}) = 0.31$
$m_i = 1$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 34.84$ $\lambda_2(\mathcal{G}) = 0.19$	$\|\Sigma_e(\mathcal{G})\|_2^2 = 35.27$ $\lambda_2(\mathcal{G}) = 0.95$

7. Concluding remarks

This work provided a characterization of how cycles impact the $\mathcal{H}_2$ performance of consensus networks. This analysis was facilitated by using an edge variant of the graph Laplacian matrix, termed the essential edge Laplacian. This matrix representation of the graph leads to a deeper insight of the role cycles play for certain algebraic properties. When applied to the corresponding edge agreement problem, the role of cycles were related to the performance of the system. In particular, the purely combinatorial property of cycle lengths and cycle correlations were shown to directly impact the $\mathcal{H}_2$ performance of the system.

The analytic results were then used to formulate an optimization problem for the design of consensus networks. Using an ℓ_1-relaxation, the problem was transformed into a semi-definite program. This relaxation turns out to be very sensitive to the weighting function used on the edges. This provides an important tuning parameter for design of these systems. Another advantage of this representation is additional performance parameters are easily embedded into the program. This was demonstrated by augmenting the program with a algebraic connectivity maximization objective. The results were demonstrated via some numerical simulations.

References

[1] M. Mesbahi, M. Egerstedt, Graph Theoretic Methods in Multiagent Networks, Princeton University Press, Princeton, NJ, 2010.
[2] J.A. Fax, R.M. Murray, Information flow and cooperative control of vehicle formations, IEEE Transactions on Automatic Control 49 (9) (2004) 1465–1476.
[3] P. Giordano, A. Franchi, C. Secchi, H. Bülthoff, Bilateral teleoperation of groups of UAVs with decentralized connectivity maintenance, in: Proceedings of Robotics: Science and Systems, Los Angeles, CA, 2011.
[4] A. Nedic, A. Ozdaglar, Distributed subgradient methods for multi-agent optimization, IEEE Transactions on Automatic Control 54 (1) (2009) 48–61.
[5] R. Olfati-Saber, Distributed Kalman filter with embedded consensus filters, in: Proceedings of the 44th IEEE Conference on Decision and Control, IEEE, 2005, pp. 8179–8184.
[6] R. Olfati-Saber, J. Shamma, Consensus filters for sensor networks and distributed sensor fusion, in: Proceedings of the 44th IEEE Conference on Decision and Control, 2005, pp. 6698–6703.
[7] M. Fiedler, Algebraic connectivity of graphs, Czechoslovak Mathematical Journal 23 (98) (1973) 298–305.
[8] Y. Hatano, M. Mesbahi, Agreement over random networks, IEEE Transactions on Automatic Control 50 (11) (2005) 1867–1872.
[9] R. Olfati-Saber, R.M. Murray, Consensus problems in networks of agents with switching topology and time-delays, IEEE Transactions on Automatic Control 49 (9) (2004) 1520–1533.

[10] G. Parlangeli, G. Notarstefano, On the reachability and observability of path and cycle graphs, IEEE Transactions on Automatic Control 57 (3) (2012) 743–748.
[11] A. Rahmani, M. Ji, M. Mesbahi, M. Egerstedt, Controllability of multiagent systems: from a graph-theoretic perspective, SIAM Journal on Control and Optimization 48 (1) (2008) 162–186.
[12] B. Briegel, D. Zelazo, M. Bürger, F. Allgöwer, On the zeros of consensus networks, in: Proc. 50th IEEE Conference on Decision and Control, Orlando, FL, 2011, pp. 1890–1895.
[13] S. Tonetti, R.M. Murray, Limits on the network sensitivity function for homogeneous multi-agent systems on a graph, in: American Control Conference, ACC, 2010, pp. 3217–3222.
[14] M.-G. Yoon, K. Tsumura, Transfer function representation of cyclic consensus systems, Automatica 47 (9) (2011) 1974–1982.
[15] S.P. Boyd, Convex optimization of graph Laplacian eigenvalues, in: International Congress of Mathematicians, Vol. 3, Madrid, Spain, 2006, pp. 1311–1310.
[16] S.P. Boyd, A. Ghosh, Growing well-connected graphs, in: Proceedings of the 45th IEEE Conference on Decision & Control, San Diego, 2006, pp. 6605–6611.
[17] L. Xiao, S.P. Boyd, S. Kim, Distributed average consensus with least-mean-square deviation, Journal of Parallel and Distributed Computing 67 (1) (2007) 33–46.
[18] R. Dai, M. Mesbahi, Optimal topology design for dynamic networks, in: 50th IEEE Conference on Decision and Control and European Control Conference, CDC 2011, IEEE, Orlando, FL, 2011, pp. 1280–1285.
[19] Y. Kim, M. Mesbahi, On maximizing the second smallest eigenvalue of a state-dependent graph Laplacian, IEEE Transactions on Automatic Control 51 (1) (2006) 116–120.
[20] A. Das, Y. Hatano, M. Mesbahi, Agreement over noisy networks, IET Control Theory & Applications 4 (11) (2010) 2416.
[21] Y. Hatano, M. Mesbahi, A. Das, Agreement in presence of noise: pseudogradients on random geometric networks, in: 44th IEEE Conference on Decision and Control, Seville, Spain, 2005, pp. 6382–6387.
[22] F. Lin, M. Fardad, M.R. Jovanovic, Algorithms for leader selection in large dynamical networks: noise-corrupted leaders, in: Proc. 50th IEEE Conference on Decision and Control, Orlando, FL, 2011.
[23] B. Bamieh, M.R. Jovanovic, P. Mitra, S. Patterson, Coherence in large-scale networks: dimension-dependent limitations of local feedback, IEEE Transactions on Automatic Control 57 (9) (2012) 2235–2249.
[24] S. Patterson, B. Bamieh, Network coherence in fractal graphs, in: Proc. 50th IEEE Conference on Decision and Control, Orlando, FL, 2011, pp. 6445–6450.
[25] G.F. Young, L. Scardovi, N.E. Leonard, Robustness of noisy consensus dynamics with directed communication, in: American Control Conference, ACC, 2010, Baltimore, MD, 2010.
[26] G.F. Young, L. Scardovi, N.E. Leonard, Rearranging trees for robust consensus, in: IEEE Conference on Decision and Control and European Control Conference, IEEE, 2011, pp. 1000–1005.
[27] L. Scardovi, M. Arcak, E.D. Sontag, Synchronization of interconnected systems with applications to biochemical networks: an input-output approach, IEEE Transactions on Automatic Control 55 (6) (2010) 1367–1379.
[28] D. Zelazo, M. Mesbahi, Edge agreement: graph-theoretic performance bounds and passivity analysis, IEEE Transactions on Automatic Control 56 (3) (2011) 544–555.
[29] C. Godsil, G. Royle, Algebraic Graph Theory, Springer, 2009.
[30] E.J. Candès, J.K. Romberg, T. Tao, Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information, IEEE Transactions on Information Theory 52 (2) (2006) 489–509.
[31] D.L. Donoho, Compressed sensing, IEEE Transactions on Information Theory 52 (4) (2006) 1289–1306.
[32] E.J. Candes, M.B. Wakin, S. Boyd, Enhancing sparsity by reweighted ℓ_1 minimization, Journal of Fourier Analysis and Applications 14 (5) (2008) 877–905.
[33] F. Lin, M. Fardad, M.R. Jovanović, Design of optimal sparse feedback gains via the alternating direction method of multipliers, IEEE Transactions on Automatic Control (2011) 1–32 (submitted for publication).
[34] S. Schuler, P. Li, J. Lam, F. Allgöwer, Design of structured dynamic output-feedback controllers for interconnected systems, International Journal of Control 84 (12) (2011) 2081–2091.
[35] R.T. Rockafellar, Network Flows and Monotropic Optimization, John Wiley, New York, NY, 1984.
[36] D. Zelazo, M. Mesbahi, Graph-theoretic analysis and synthesis of relative sensing networks, IEEE Transactions on Automatic Control 56 (5) (2011) 971–982.
[37] G. Dullerud, F. Paganini, A Course in Robust Control Theory: A Convex Approach, Springer-Verlag, New York, 2000.
[38] K. Petersen, M. Pedersen, The matrix cookbook, Technical University of Denmark, 2008.
[39] I. Holyer, The NP-completeness of some edge-partition problems, SIAM Journal on Computing 10 (4) (1981) 713–717.
[40] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons Ltd., West Sussex, England, 1986.
[41] M. Fazel, H. Hindi, S. Boyd, A rank minimization heuristic with application to minimum order system approximation, in: Proc. American Control Conference, ACC, 2001, pp. 4734–4739.
[42] M. Fazel, Matrix rank minimization with applications, Ph.D. Thesis, Department of Electrical Engineering, Stanford University (2002).
[43] A. Caprara, Packing cycles in undirected graphs, Journal of Algorithms 48 (1) (2003) 239–256.
[44] J. Löfberg, YALMIP: a toolbox for modeling and optimization in Matlab, in: Proc. CACSD Conf., 2004, pp. 284–289.
[45] J.F. Sturm, Using SeDuMi, Optimization Methods and Software 11–12 (1–4) (1999) 625–653.