Distributed Cycle Detection and Removal

Gabriele Oliva\textsuperscript{1}, Roberto Setola\textsuperscript{1}, Luigi Glielmo\textsuperscript{2}, and Christoforos N. Hadjicostis\textsuperscript{3}

Abstract—In this paper we provide distributed algorithms to detect and remove cycles in a directed relational graph by exploiting the underlying undirected communication graph; the relational graph models a relation among the agents, e.g., a pairwise ordering, while the communication graph captures how information can be shared among them. Specifically, we provide a synchronous distributed algorithm to detect cycles in the relational graph, by exploiting the fact that nodes with zero indegree or zero out-degree can not be part of a cycle, and can be iteratively removed without creating new loops in the relational topology. The proposed algorithm considerably improves transmission efficiency (the number of messages and bandwidth required) compared to the state of the art. We demonstrate that the problem of making the relational graph acyclic by swapping the orientation of a minimum number of edges is NP-Complete and APX-Hard; for this reason, we develop an efficient, yet suboptimal, distributed algorithm to remove the cycles by swapping the direction of some of the links. The methodology provided in this paper finds application in several distributed control problems where the agents must be interconnected via a directed acyclic graph, such as cluster consensus, formation control or multiple leader election. Extensive numerical analysis emphasizes the effectiveness of the proposed solution with respect to the state of the art.

Keywords: Distributed Systems, Cycle Detection, Cycle Removal, Deadlocks, Minimum Feedback Arc Set Problem.

I. INTRODUCTION

Cycles in networks are either a blessing or a curse. On one hand, from a structural point of view, cycles imply the existence of redundant links and paths, that can be used to guarantee connectivity among the nodes in the event of signal fading, congestions or failures. On the other hand, the presence of a cycle might indicate a violation of the logical or relational structure that underlies the interaction among the nodes; this is well represented by Escher’s paintings or by the well known causal cycle paradox [1], where a future event is the cause of a past event, which in turn is the cause of the future event. Further issues that can be modeled by a cycle include, among others, deadlocks [2]–[5], stalls in Petri nets [6], or inconsistencies in network inference techniques such as Bayesian Belief Networks [7].

Cycles can also arise due to errors or due to malicious attacks aiming at steering the system towards an inconsistent state. Consider, for instance, a network of agents, each with a task, and suppose that neighboring agents can not execute their tasks at the same time (e.g., they share a common tool or resource). In this case, there is a need to define a priority for each pair of neighboring agents. Notice that the resulting directed graph must be acyclic to avoid deadlocks.

Another relevant example is when the orientation of the links represents the routing of messages from an agent to another. Also in this case, we need to guarantee that the relational topology is acyclic; otherwise, the messages may persist indefinitely in the system. Indeed, when the directed links represent a flow of resources, such as water or gas, it might be desirable to get rid of loops to prevent dangerous or inconsistent configurations.

Directed acyclic graphs find several applications in the field of distributed control systems: in [9] the authors provide an algorithm to let a set of agents pursue each other; in [8] a decentralized observer is developed; in [13] the authors provide a framework to let a secondary team of mobile robots recharge a primary team, in order to guarantee their continuous operation; in [10] directed acyclic graphs are shown to guarantee the achievement of cluster consensus control, regardless of the magnitudes of the coupling strengths among the agents, while in [11] such results are extended to second-order agent systems; in [12] (and references therein) the problem of distributed formation control is tackled by resorting to acyclic formations, arguing that, in the presence of cycles, the formation may fail to converge to its desired shape.

Notice that, in several of the aforementioned situations, there might be the need to provide a directed acyclic graph that preserves, as much as possible, the relations specified in the initial topology, where the cycles are present. Consider, for instance, a set of agents, interconnected by a directed graph that represents locally desirable leader/follower relations. Suppose the agents want to avoid logical inconsistencies due to loops in the relational topology, and they need to elect one or more leaders (e.g., the leaders are not following any other node), so that structural controllability is achieved for the network. In this case, building an acyclic graph ex novo is not desirable; an alternative strategy might be to operate small changes in the...
orientation of some of the links, while leaving the remaining links unchanged.

Motivated by the above considerations, in this paper, we develop distributed algorithms to detect cycles in directed graphs and to remove them by swapping the direction of a few links, attempting to preserve the original orientation of as many edges as possible. Specifically, we consider a network of \( n \) agents, interconnected at two different levels, as shown in Figure 1. From a relational point of view, the agents are interconnected via the connected directed graph \( G_r \) (blue dashed lines in Figure 1). Moreover, they are also connected (as typically done in cyber-physical systems) via a communication layer (black solid lines in Figure 1), represented by a connected undirected graph \( G_c \) with the same structure as \( G_r \).

In this paper we provide distributed algorithms to let the agents detect and remove cycles in the relational layer \( G_r \), by exchanging information over the communication network \( G_c \). Moreover, to remove such cycles, we assume that the agents are allowed to swap the orientation of the links in \( G_r \). Notice that we swap the direction of the links, instead of simply removing them, because we want to guarantee that the graphs remain connected or that there is limited degradation in the network transportation capacity, given also the possibly sparse nature of the interaction among the agents.

A. Related Work

Several approaches are available in the literature to detect cycles in directed graphs (see [14] for a recent survey). Recent works on the topic include distributed algorithms that aim at maintaining an acyclic graph when new links are added to an initially acyclic graph [15], centralized approaches based on Ordinary Differential Equations to detect deadlocks in Petri nets [6], and centralized methodologies to resolve deadlocks in databases via soft computing [5].

Traditional methods rely on depth-first searches (DFS) [16], [18], exploiting the fact that a graph has a cycle if and only if the DFS finds a so-called back edge. A back edge is found by storing the list of visited nodes and checking for repeated entries in the list. This approach is quite efficient in a centralized context and requires \( O(n) \) steps to terminate, but it has several drawbacks when applied in a distributed manner:

i) The size of the messages exchanged within distributed DFS, in fact, is \( O(n \log n) \) bits, as the nodes append their identifier to the message before forwarding it. This requirement can be prohibitive in large networks because of the need to have high memory resources at each node and high bandwidth occupation during the transmission of the messages.

ii) Another critical aspect is that distributed DFS requires nodes able to communicate on a point-to-point basis, which is a strong assumption in the context of distributed algorithms. In a completely distributed context, in fact, it might be preferable to have nodes that communicate by broadcasting their messages to neighbors, without addressing them directly (this is particularly advantageous in wireless settings where a single transmission by a node is simultaneously received by all of its neighbors, or in cases where the communication graph –unlike the case here– might be directed).

iii) In distributed DFS, a leader has to be elected, and the leader initiates the DFS by sending its identifier to one of its not-yet-visited neighbors; this neighbor does the same, and so on until all reachable nodes are visited, or a cycle is found. To apply this approach to the problem discussed in this paper, however, we need further leader election procedures, as the directed relational graph has no guarantee to be strongly connected (hence there might be no way to reach all the nodes with a single DFS). In this case, to ensure that all cycles are found, there might be the need to repeatedly elect a new leader among the nodes that have not yet been visited, until all nodes are visited. This can be done by exploiting the underlying undirected communication topology, for instance via max-consensus [20].

In [17] Rocha and Thatte consider the detection of cycles and provide a synchronous distributed algorithm, in which the nodes broadcast their messages. This algorithm requires \( \delta \) steps to terminate, where \( \delta \) is the network diameter (if \( \delta \) is unknown we can assume \( O(n) \) steps are required). The agents, however, transmit messages of \( O(n \log n) \) bits (because messages contain the list of visited nodes as in standard distributed DFS), and the bandwidth occupation is worsened by the broadcasting approach with respect to distributed DFS. In this approach, however, there is no need to elect leaders, saving some computational time.

In [3], [4] a centralized approach to detect deadlocks is provided, where first the size of the graph is reduced by iterated removal of nodes with zero in-degree or out-degree, and then DFS is used to find cycles.

B. Contribution

In this paper, we provide a synchronous distributed algorithm to detect cycles in a directed graph, namely the Distributed Cycle Detection (DCD) algorithm. Rather than exploring the network and appending identifiers to messages, the algorithm is based on the property that nodes with zero in-degree or zero out-degree cannot be part of a cycle, as noted in [3], [4]. Within the proposed algorithm, therefore, the agents with zero in-degree or out-degree are labeled as not belonging to a cycle, and such agents do not contribute anymore to the search of cycles. Such a procedure is iterated over the remaining nodes, until no more nodes can be removed.

Specifically, we prove in Section III that the set \( V_D \) of nodes remaining, after the iterated removal of nodes with zero in- or out-degree, contains the set \( V_C \) of nodes in a cycle (or cycles), and that \( V_D \) is nonempty if and only if the network contains at least a cycle. As a result, we develop a distributed local cycle detection algorithm, where each agent detects whether or not it belongs to \( V_D \), but has no information about other agents. To obtain global information (i.e., to detect the existence of a cycle), a max-consensus procedure [20] is executed to let all the agents agree on the existence of cycles in the network (i.e., to inform also the nodes that are not in \( V_D \)). The overall approach requires \( O(n) \) steps to terminate.
and messages of size $O(\log n)$ bits, thus reducing to a relevant extent the memory and bandwidth requirements with respect to approaches based on DFS.

We notice that, in order to remove cycles, there is no need to explicitly list them, as done in previous work. Instead, it is sufficient to find a mechanism to swap the links of nodes that belong to a cycle, assuming this is possible in the underlying problem.

Based on the above intuition and the DCD algorithm, we develop a distributed algorithm, namely Distributed Cycle Removal (DCR), that removes cycles by swapping the orientation of some of the links in $G_r$. The algorithm has the same memory and bandwidth requirements as the DCD algorithm. During its initialization, the local cycle detection procedure is executed. Then, each round of the algorithm alternates between two phases: in the first phase, a node in $V_D$ is selected and all its incoming links are swapped; in the second phase, the local DCD Algorithm is executed again, updating $V_D$. We prove in Section V that the size of $V_D$ is monotonically reduced at each round, and that this procedure does not generate new cycles; hence, $G_r$ becomes acyclic in $O(n^2)$ steps ($O(n)$ rounds, each requiring $O(n)$ steps).

Notice that the proposed algorithm does not guarantee to find the minimum number of edges to be swapped to make the resulting graph acyclic. We study the complexity of this problem, which we refer to as the minimum swap feedback arc problem, and we prove in Section IV it is NP-Complete and APX-Hard, by reducing in polynomial time the problem to a minimum feedback arc problem [23] (i.e., finding a minimum number of edges to be removed to make the resulting graph acyclic).

C. Outline of the paper

Section II provides some preliminary definitions; Section III discusses the proposed distributed algorithms to detect the presence of cycles; Section IV focuses on the computational complexity of swapping a minimum number of links that makes the resulting graph acyclic; Section V introduces the proposed distributed algorithm to remove cycles in the relational topology by swapping some of the links; Section VI is devoted to a simulation analysis of the characteristics of the proposed algorithms, and a comparison with the state of the art; some conclusive remarks and future directions are collected in Section VII.

II. Preliminaries

Let us denote the cardinality of a set $S$ by $|S|$.

A. Approximability of NP Optimization Problems

Given an optimization problem and an approximated algorithm to solve that problem, the approximation ratio $c > 1$ is the ratio between the cost of the solution found and the cost of the optimal one. An NP optimization problem belongs to the Polynomial Time Approximation Scheme (PTAS) class if for any $c > 1$ there is a polynomial time algorithm whose approximation ratio is guaranteed to be less than or equal to $c$. An NP optimization problem is said to belong to the APX class if there is a polynomial time approximated algorithm with approximation ratio upper-bounded by a constant. Notice that, unless $P = NP$, it can be shown that there are problems in APX that are not in PTAS, i.e., problems that can be approximated for some approximation ratio, but not for any approximation ratio. The APX-Hard approximation class is the subset of APX problems for which it is possible to demonstrate that, unless $P = NP$, no polynomial time approximation algorithm exists whose approximation ratio is below a constant $c > 1$.

B. Graphs: General Definitions

Let $G = (V, E)$ denote a graph with a finite number of nodes $v_i \in V$ and $e$ edges $(v_i, v_j) \in E$ from node $v_i$ to node $v_j$. A graph is said to be undirected if $(v_i, v_j) \in E$ whenever $(v_j, v_i) \in E$, and it is said to be directed otherwise. A graph is connected if each node can be reached by each other node by means of the links in the graph, without necessarily respecting the orientation of the links, while it is strongly connected if the orientation of the links is considered.

The diameter $\delta$ of a graph $G$ is the length of the maximum among the minimum paths between all pairs of nodes. For a (strongly) connected (directed) graph, it holds $\delta \leq n - 1$, where $n$ is the number of nodes in the graph.

A directed graph is a strictly directed graph if $(v_i, v_j) \notin E$ whenever $(v_i, v_j) \in E$. A directed graph $G' = (V, E')$ is an orientation of an undirected graph $G = (V, E)$ if for each $(v_i, v_j), (v_j, v_i) \in E$ either $(v_i, v_j) \in E'$ or $(v_j, v_i) \in E'$. It follows that an orientation is always a strictly directed graph. We say a set of edges is consecutive if the edges are in a sequence, e.g., $\{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$ (a single edge is consecutive).

Let the in-neighborhood $N_i^{in}$ of a node $i$ over a directed graph $G = (V, E)$ be the set of nodes $\{v_j | (v_j, v_i) \in E\}$, while the out-neighborhood $N_i^{out}$ is the set of nodes $\{v_j | (v_i, v_j) \notin E\}$. For undirected graphs $N_i^{in} = N_i^{out} = N_i$, where $N_i$ is simply the neighborhood of node $v_i$.

Let the in-degree $d_i^{in}$ of a node $v_i$ be the number of its incoming edges, i.e., $d_i^{in} = |N_i^{in}|$, while the out-degree $d_i^{out}$ is the number of its outgoing edges, i.e., $d_i^{out} = |N_i^{out}|$. For undirected graphs, it always holds that $d_i^{in} = d_i^{out}$, and in this case it is simply referred to as the degree $d_i$ of node $v_i$.

C. Graphs: Cycle-related Definitions

A cycle is a cyclic sequence of edges such as $\{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$. A graph is acyclic if it has no cycles. We let $V_C \subseteq V$ be the set of nodes belonging to at least one cycle.

Given a directed graph $G = (V, E)$ and a set $E_m \subseteq E$, let $G' = (V, E \setminus E_m)$ be the subgraph, i.e., the graph obtained by removing the links in $E_m$. Let us define $E_m' = \{(v_i, v_j) : (v_j, v_i) \in E_m\}$
as the swap of \( E_m \), i.e., the set of links connecting the endpoints of the links in \( E_m \), but with the opposite orientation, and let

\[
G'' = \{V, (E \cup E'_m) \setminus E_m\},
\]

be the swapped graph, i.e., the graph obtained from \( G \) by substituting the links in \( E_m \) with the links in \( E'_m \).

A set \( E_m \subseteq E \) is a feedback arc set (FAS) for a directed or undirected graph \( G \) if the corresponding subtract graph \( G' \) is acyclic. In the case of a strictly directed graph, a set \( E_m \subseteq E \) is a swap feedback arc set (SAS) if the corresponding swapped graph \( G'' \) is acyclic.

We say a node \( v_i \in V \) in a directed graph \( G \) is extremal if \( d_i^{\text{in}} = 0 \) or \( d_i^{\text{out}} = 0 \). Moreover, a node \( v_i \in V \) in a directed graph \( G \) is out-of-loop if it is extremal, or if it becomes extremal by iteratively removing extremal nodes (and related edges); we call \( V_E \) the set of out-of-loop nodes and \( V_D = \overline{V_E} = V \setminus V_E \) its complement.

**D. Assumptions**

Let us make the following technical assumptions:

A1) the agents communicate synchronously over \( G_C \) on a discrete-time basis, broadcasting their messages to all their neighbors.

A2) each node \( v_i \in V \) is aware of its neighbors and the orientation of the links towards or from its neighbors in \( G_r \).

A3) each agent \( v_i \) has a unique identifier, which for simplicity we denote by \( i \);

A4) the agents know an upper bound \( \bar{n} \) on the number \( n \) of agents in the network, which is the same for each of them.

**E. Max-Consensus Algorithm**

Let us discuss the distributed max-consensus algorithm.

Suppose each node \( v_i \) in a graph \( G \) represents an agent with an initial condition \( x_{i0} \in \mathbb{R} \). The max-consensus algorithm lets all agents calculate the maximum among the initial conditions. This is achieved via the synchronous update rule

\[
x_i(k + 1) = \max_{j \in N_i^{\text{out}}(i)} \{x_j(k)\}. \tag{1}
\]

Assuming the graph is connected (in the undirected graph case), or strongly connected (in the directed graph case), the problem is known to have a solution in finite time [20], and specifically, in no more than \( \delta \) steps, where \( \delta \) is the diameter of the network. Since the diameter is in general unknown, we assume the agents execute the update rule for \( \bar{n} \) steps (see Assumption A4).

We briefly denote the above procedure for agent \( i \) using the notation

\[
x_i = \text{max-consensus}_i(x_{10}, \ldots, x_{n0}).
\]

**Algorithm 1:** Distributed Local Cycle Detection (DLCD) for node \( v_i \)

1: \( \triangleright \) Initialization
2: \( t = 0; \)
3: out-of-loop \(_i = 0; \)
4: in = \( d_i^{\text{in}}; \)
5: out = \( d_i^{\text{out}}; \)
6: \( \triangleright \) Check if \( v_i \) is out-of-loop
7: \( \textbf{while} t < \bar{n} \) \( \textbf{do} \)
8: \( \text{if} \ \text{in} == 0 \) or \( \text{out} == 0 \) \( \text{then} \)
9: \( \text{out-of-loop}_i = 1; \)
10: broadcast \( i \) to all neighbors in \( G_C; \)
11: \( \textbf{break}; \)
12: \( \textbf{end if} \)
13: update in and out based on received messages;
14: \( t = t + 1; \)
15: \( \textbf{end while} \)
16: \( \textbf{return} \ \text{out-of-loop}_i \)

**Algorithm 2:** Distributed Cycle Detection (DCD) for node \( v_i \)

1: \( \triangleright \) Check if \( v_i \) is out-of-loop;
2: out-of-loop \(_i = \) DLCD;
3: \( \triangleright \) Check if \( G_r \) contains a cycle
4: \( \zeta_i = 1 - \text{out-of-loop}_i; \)
5: has-cycle = \text{max-consensus}_i(\zeta_1, \ldots, \zeta_n); \)
6: \( \textbf{return} \ \text{has-cycle} \)

**III. DISTRIBUTED CYCLE DETECTION**

In this section, we aim at identifying the nodes in \( V_C \), which is the set of nodes belonging to at least one cycle. To this end, we provide two distributed algorithms to detect cycles locally (Algorithm 1) and globally (Algorithm 2), which stem from the following results. Specifically, we show next that \( V_C \) is a subset of the set \( V_D \) of nodes that are not out-of-loop, i.e., such that they are not extremal, nor become extremal by iteratively removing extremal nodes\(^1\) (and related edges).

**Proposition 1:** Consider a strictly directed graph \( G_r = \{V, E_r\} \). Then \( V_C \subseteq V_D \).

**Proof:** A node \( v_i \) belonging to a cycle has necessarily at least one incoming edge and one outgoing edge, hence \( d_i^{\text{in}} \neq 0 \) and \( d_i^{\text{out}} \neq 0 \), and so does at least one of its in-neighbors and at least one of its out-neighbors. It can be noted, therefore, that by iteratively removing any extremal node (and its incident edges) from \( G_r \), we can never remove nodes belonging to a cycle in \( G_r \), i.e., \( V_C \cap V_E = \emptyset \), which is the thesis. \( \blacksquare \)

Based on the above Proposition, we develop a distributed local algorithm, namely Distributed Local Cycle Detection (DLCD) Algorithm, to identify the presence of cycles, i.e., such that an agent is able to understand whether it belongs to a set of nodes containing a cycle.

\(^1\)Recall that a node is extremal if its in- or out-degree is zero.
At the beginning, all nodes are active and, at each round of DLCD, the active nodes check their in- and out-degree; if one of them is equal to zero for a node \( v_i \), it labels itself as out-of-loop and it broadcasts its identifier \( i \) to all its neighbors; after that it becomes inactive, i.e., it does not contribute further to DLCD algorithm (it communicates again during the max-consensus procedure within the DCD Algorithm, as discussed later). Based on the possible reception of messages that declare that a neighbor is out-of-loop, each node updates at each step its in- and out-degree.

Moreover, each node keeps track of the number \( t \) of rounds done, and it becomes inactive when \( t \geq \hat{n} \); in this case, if the node has both nonzero in-degree and out-degree it belongs to \( V_D \). The algorithm terminates by returning the Boolean variable out-of-loop \( \zeta_i \), which is one if node \( v_i \) belongs to \( V\backslash V_D \) and zero otherwise.

The Distributed Cycle Detection (DCD) Algorithm (Algorithm 2) is used to let each node detect cycles; it is composed of an execution of DLCD and a max-consensus algorithm where the agents select an initial condition

\[
\zeta_i = 1 - \text{out-of-loop}_i,
\]

which is a Boolean variable such that \( \zeta_i = 1 \) if \( v_i \) is in \( V_D \) and \( \zeta_i = 0 \) otherwise. By calculating the maximum among the \( \zeta_i \), all nodes are able to detect the existence of a cycle.

Notice that \( V_D \) does not necessarily coincide with the set \( V_C \), which contains only the nodes in a cycle, and it is possible that \( V_C \) is strictly contained in \( V_D \) (an example is provided below). However, the following result justifies the adoption of Algorithm 1 to detect the presence of a cycle.

**Proposition 2:** \( V_D \neq \emptyset \) if and only if \( V_C \neq \emptyset \).

**Proof:** \( V_C \neq \emptyset \Rightarrow V_D \neq \emptyset \) trivially. Conversely, if \( V_D \neq \emptyset \) then one can construct at least one cycle as follows. We start by choosing any node in \( V_D \), then we select one of its out-neighbors. Iterating this procedure, each time moving to an out-neighbor in \( V_D \), we are guaranteed to end up visiting an already visited node, because each node in \( V_D \) has at least one in- and one out-neighbor in \( V_D \), and the set \( V_D \) is finite.

**Remark 1:** Proposition 1 implies that, after all out-of-loop nodes \( v_i \) have been iteratively removed, the remaining nodes in \( V_D \), if any, must belong to at least one cycle or to a path connecting cycles (see an example in Figure 2). In other words, the set of non-out-of-loop nodes found by DCD may be larger than the set of nodes belonging to a cycle. In view of this, an agent in \( V_D \) is not able to determine whether it actually belongs to a cycle.

Let us conclude the section discussing the computational properties of the DCD Algorithm.

**Remark 2:** The DLCD and DCD algorithms terminate in \( O(n) \) steps and require \( O(\log n) \) bits of bandwidth per link per step and \( O(d_i \log n) \) bits of memory at each agent, where \( d_i \) is the degree of node \( v_i \) over \( G_c \). It should be noted, in fact, that the messages exchanged at each step have size \( O(\log n) \) bits, as each contains one identifier of a node (which requires \( O(\log n) \) bits), or a Boolean (i.e., the variables exchanged in the max-consensus). This implies that, at each step, the bandwidth used for each link is \( O(\log n) \) bits. Moreover, the memory required is \( O(d_i \log n) \) bits, as each node needs a buffer to store the messages received by each neighbor over \( G_c \) at each step. As for the number of steps required to terminate, it should be noted that \( \hat{n} \) steps are required for the iterative removal of nodes and \( \hat{n} \) additional steps for the max-consensus, hence both DLCD and DCD require \( O(n) \) steps.

**IV. Minimum Swap Feedback Arc Set**

Before discussing, in the next section, a distributed algorithm to remove cycles in a strictly directed graph by swapping some of its edges, we define the Minimum Swap Feedback Arc Set Problem (mSAS), and we characterize its computational complexity. Over a strictly directed graph \( G \), the problem consists of finding the SAS \( E_m \) of minimum cardinality, such that the swapped graph obtained by swapping the orientation of the links in the SAS is acyclic. In other words, the problem we introduce in this section amounts to selecting a minimum number of edges that, if their orientation is swapped, make the resulting graph acyclic. To the best of our knowledge, mSAS is novel, thus no characterization of its complexity has been provided in the literature.

In order to determine the complexity of the problem at hand, in the remainder of this section we discuss the relationship between the above problem and a similar problem, namely Minimum Feedback Arc Set Problem (mFAS). mFAS consists of finding the FAS \( E_m \) of minimum cardinality over a directed (or undirected) graph \( G \), i.e., a minimum number of edges that, if removed, make the resulting graph acyclic. Although solvable in polynomial time for the class of planar directed graphs [21], such a problem is known to be NP-complete in the directed graph case [22] and it was included by Karp in his famous list of 21 NP-Complete problems [23].

Notice that, besides being NP-Complete, the mFAS problem is APX-Hard [24]. As demonstrated in [25], the highest known value of the approximation ratio \( c \) such that no polynomial time algorithm with smaller approximation ratio exists is 1.3606. In other words, no polynomial-time algorithm can be guaranteed to produce an edge set at most 36.06% larger than the optimal solution, although the result of the approximated algorithm can be better on particular instances.
can conclude that $E_{mSAS}$ is a minimum cardinality FAS over $G_+$.

Let the set

$$E_{mFAS} = \{(v_i, v_j) \in E : (v_i, v_j) \in E_{mSAS} \text{ or } (v_j, v_i) \in E_{mSAS}\}.$$ 

The subtract graph $G'_+\ast$ obtained by removing the edges in $E_{mSAS}$ from $G_+$ is acyclic, and if we revert all the remaining pairs $(v_i, v_{ij}), (v_j, v_j)$ to the corresponding edge $(v_i, v_j)$ of $G$ and we remove the nodes in $V_+\setminus V$, we obtain a graph $G'$ that is again acyclic, and in particular, we obtain a subtract graph of the original graph $G$ with respect to $E_{mFAS}$.

Let us show that $E_{mFAS}$ is a minimum cardinality FAS over $G$. Suppose there is a minimum cardinality FAS $E_{mFAS}'$ over $G$ with $|E_{mFAS}'| < |E_{mFAS}|$; by choosing over $G_+$ just one edge in the pair $(v_i, v_{ij}), (v_j, v_j)$ for each $(v_i, v_j) \in E_{mFAS}$ we obtain a FAS in $G_+$ with a smaller cardinality than $E_{mFAS}$, a contradiction.

Since $G_+$ and $G'$ can be obtained in polynomial time from $G$ and $G_+$ respectively, the thesis follows. 

**Corollary 1:** A minimum cardinality SAS and a minimum cardinality FAS over a graph $G$ have the same cardinality.

**Proof:** From Lemma 1, a minimum cardinality SAS with cardinality $m'$ over $G_+$ corresponds to a minimum cardinality FAS over $G_+$ with cardinality $m'$; this, in turn, induces a minimum cardinality FAS over $G$ with the same cardinality.

With a similar argument as in Lemma 1, we can conclude that a minimum cardinality SAS over $G_+$ corresponds to a minimum cardinality SAS over $G$ with cardinality $m'$.

**Remark 3:** With a similar argument as in Lemma 1, it can be shown that mSAS can be reduced in polynomial time to mFAS.

We can now state Theorem 1.

**Theorem 1:** mSAS is NP-Complete and APX-Hard.

**Proof:** The proof of NP-Completeness is a straightforward consequence of Lemma 1, and of the fact that mFAS is NP-Complete [22], [23].

Suppose that mSAS is in PTAS; hence, for each $c' > 1$ there is a polynomial time algorithm that finds a solution with cardinality $m'\ast$ such that $m' \leq c'm^\ast$, where $m^\ast$ is the cardinality of a minimum cardinality SAS. By Corollary 1, $m^\ast$ is also the cardinality of an mFAS over $G$. Notice that an SAS over $G$ is also an FAS over $G$ (i.e., if the graph is made acyclic by swapping the edges in the SAS, it is also acyclic by removing them). We conclude that, by using the hypothetical polynomial time algorithm for finding an SAS, we find an FAS whose cardinality $m''$ is

$$m'' = m' \leq c'm^\ast. \quad (2)$$

Since Ineq. (2) holds true for all $c' > 1$, we must conclude mFAS is in PTAS and thus it is not APX-Hard, a contradiction. The proof is complete.
Corollary 2: The approximation ratio \( c' \) of mSAS is greater than or equal to the approximation ratio \( c \) for mFAS, i.e.,
\[
c' \geq c \geq 1.3606.
\]

Remark 4: According to the above result, since mSAS is NP-Complete and APX-Hard for some constant \( c' \geq 1.3606 \), it is not trivial to solve it exactly, let alone in a distributed way. Notice that finding an upper bound on the approximation ratio \( c' \) of mSAS is a valuable future work direction.

In the next section, we develop a distributed algorithm to calculate an SAS with no guarantee of minimum cardinality.

V. DISTRIBUTED CYCLE REMOVAL

In this section we provide a distributed algorithm, based on DLCD, to remove the cycles in \( G_r \) by iteratively selecting a pivot node and by swapping its incoming links.

In the following, we consider a strictly directed graph \( G_r = \{V,E_r\} \) and, with a slight abuse of notation, we let \( G_r(t) = \{V(t),E_r(t)\} \) and \( V_D(t) \) be the graph and the subsets of its nodes obtained after round \( t \) while executing Algorithm DLCD within Algorithm 3, with the original graph at \( t = 0 \).

Let us state the following proposition.

Proposition 3: Provided \( V_D(t-1) \neq \emptyset \),
\[
V_D(t) \subseteq V_D(t-1)
\]
for all \( t = 1,\ldots,n \). Moreover, Algorithm 3 succeeds in restoring the acyclicity of \( G_r \) in at most \( |V_D(0)| \) rounds.

Proof: We show, equivalently, that \( V_E(t) \supset V_E(t - 1) \), provided \( V_E(t-1) \neq \emptyset \). Let \( v_j \in V_D(t-1) \subseteq V_E(t-1) \) be the pivot node chosen at time \( t \). After swapping its incoming edges it becomes an extremal node and it is removed, together with its edges, by the next DLCD. Hence its swapped (and afterwards deleted) edges cannot determine any change at nodes of \( V_E(t-1) \), so that \( V_E(t) \supset V_E(t - 1) \cup \{v_j\} \), which proves both claims.

Based on the above proposition we can provide Algorithm 3 as a distributed way to restore the acyclicity of the graph \( G_r \), by swapping the orientation of some of the links of a subset of the nodes. The algorithm alternates between a local cycle detection phase, where Algorithm 1 is used to detect the presence of cycles, and a phase where a leader is elected among the non-out-of-loop nodes; the leader then swaps all its incoming edges. The procedure is iterated until all cycles are removed.

Figure 4 shows an example of the application of Algorithm 1 and Algorithm 3 for a network with \( n = 20 \) nodes (in random position in the unit square \( [0,1] \times [0,1] \)) and \( e = 46 \) links (we connect two nodes if their distance is less than 0.35).

In the left plot we show the results of Algorithm 1: there are \( |V_D| = 14 \) non-out-of-loop nodes (the nodes in \( V_C \) are shown in light blue, and the nodes in \( V_D \setminus V_C \) are shown in black) and \( |V \setminus V_D| = 6 \) out-of-loop nodes (in red). Notice that, in the above example, we have \( |V_D \setminus V_C| = 3 \) nodes that are non-out-of-loop but do not belong to a cycle. In the right plot we show the result of Algorithm 3. The algorithm requires 4 rounds to remove all the cycles, and the swapped links are 11 (in magenta).

Let us provide an upper bound \( \theta \) on the number of links swapped by Algorithm 3.

Proposition 4: Algorithm 3 swaps at most
\[
\theta = d_{\max}|V_D(0)|
\]
links, where \( d_{\max} \) is the maximum among the degrees of the nodes over \( G_c \).

Proof: By running the DCD algorithm for the first time within the DCR algorithm, the agents are able to detect whether or not they belong to \( V_D(0) \). Then, the DCR Al-

Algorithm 3: Distributed Cycle Removal (DCR) for node \( v_i \)

1: \( \triangleright \) Initialization
2: out-of-loop_\( i \) = DLCD;
3: pivot = \( \infty \);
4: \( \triangleright \) Check for cycles and rewire edges until \( V_D = \emptyset \)
5: while pivot > 0 do
6: \( \triangleright \) select a pivot among nodes with out-of-loop_\( j \) = 0
7: if out-of-loop_\( j \) == 0 then
8: \( \triangleright \) identify_\( j \) = \( i \);
9: else
10: \( \triangleright \) identify_\( j \) = 0;
11: end if
12: pivot = max-consensus(identify_\( 1 \),\ldots,identify_\( n \));
13: \( \triangleright \) if \( v_i \) is the pivot, swap its incoming edges
14: if pivot == \( i \) then
15: \( \triangleright \) replace links \((v_j,v_i)\) in \( G_r \) with \((v_i,v_j)\) for all \( j \);
16: else
17: \( \triangleright \) swap links towards the pivot, if a pivot is found
18: if pivot > 0 then
19: \( \triangleright \) replace links \((v_i,v_{\text{pivot}})\) in \( G_r \) with \((v_{\text{pivot}},v_i)\);
20: end if
21: end if
22: \( \triangleright \) Update cycle information
23: out-of-loop_\( i \) = DLCD;
24: end while
25: return

Fig. 4. Example of distributed cycle detection by means of Algorithm 1 (left plot) and distributed cycle removal by means of Algorithm 3 (right plot). In the left plot, the nodes in \( V_C \) are marked in light blue, the nodes in \( V_D \setminus V_C \) are shown in black, and the nodes in \( V_E \) = \( V \setminus V_D \) are reported in red. In the right plot \( V_D = \emptyset \) and the graph is acyclic; the swapped links are reported in purple.
algorithm iteratively removes from \( V_D(0) \) at least one node at each round; moreover, at most \( d_{\max} \) links are swapped for each node. The thesis follows.

Let us discuss the computational properties of the DCR algorithm.

**Proposition 5.** Algorithm 3 terminates in \( O(n^2) \) steps and requires \( O(\log n) \) bits of bandwidth per link per step and \( O(d_i \log n) \) bits of memory at each agent \( i \), where \( d_i \) is the degree of node \( v_i \) over the communication graph \( G_C \).

**Proof:** By Proposition 3, the set \( V_D \) of nodes found by the DLCD algorithm is reduced by at least one node at each round and \( |V_D| \leq n \) holds, hence the algorithm terminates in at most \( n \) rounds. Each round is composed of an execution of the DLCD algorithm and a max-consensus procedure; hence, as noted in Remark 2, the algorithm terminates in \( O(n^2) \) steps. Moreover, the memory and bandwidth requirements are the same as Algorithm DLCD, since the agents simply execute max-consensus procedures and removal of nodes with zero in- or out-degree, as discussed in Remark 2.

**Remark 5.** Algorithm 3 does not specify a criterion to choose the pivot node (the node in \( V_D \) with maximum identifier was selected at each round, but any other choice of a node in \( V_D \) would also work). For instance, if we aim at swapping the links of just a few nodes (i.e., if we aim at preserving, as much as possible, the initial orientation of the links), we might want to select first the nodes with greater in-degree in \( V_D \); we show in the next section that this strategy is effective in reducing the number of rounds (and, consequently, the number of swapped edges). In this case we need to use max-consensus to select the node in \( V_D \) with maximum in-degree, at the cost of an additional max-consensus procedure at each round of Algorithm 3; hence, the total number of steps is increased. Notice, however, that since the problem at hand is APX-Hard there is no way to select the pivot nodes in order to guarantee that the resulting approximation ratio is less than \( c' \geq 1.3606 \).

To conclude the section, let us show in Figure 5 an instance where the DCR algorithm may behave in a very suboptimal way. Specifically, the graph is composed of \( n \) nodes (in the figure, we chose \( n = 8 \)) connected via a directed cycle over all the nodes (the links are oriented from \( v_1 \) to \( v_{i+1} \)), and each node except \( v_1 \) and \( v_n \) has an additional edge towards \( v_n \). In this case, the optimal solution is to select node \( v_1 \) as pivot, thus resulting in just one swap (see the right plot in Figure 5). It is, however, possible, to select other nodes, thus resulting in one of the two situations depicted in the left and central plots in Figure 5: in central plot, just one node (i.e., \( v_n \)) is selected as pivot, but \( n - 2 \) links are swapped; in the left plot, instead, \( n - 2 \) nodes are selected as pivots and, again, \( n - 2 \) links are swapped.

**VI. SIMULATIONS**

Figure 6 shows the results of a comparison of the proposed DCD algorithm against the distributed DFS and against the algorithm from Rocha and Thatte [17]. In particular, we consider nodes in random position in the unit square \([0, 1] \times [0, 1] \) (\( n = 20 \) in the upper plots and \( n = 100 \) in the lower plots), and we connect them with random directions if their distance is less than a threshold \( \rho \). We plot, for each of the algorithms and for several values of \( \rho \), the number of messages exchanged (left column) and the total number of bits exchanged (central column) by all agents. For simplicity we assume \( n = n \) and we neglect in all three algorithms the final steps required to diffuse information on the existence of cycles among all agents. For each choice of \( \rho \) we generate \( m = 100 \) graphs and we run each algorithm, reporting the mean values and the standard deviations. Moreover, we report the fraction of nodes in \( V_D \setminus V_C \), i.e., the number of nodes found by the DCD algorithm that are not in a cycle but are labeled as belonging to \( V_D \). This can be considered as a measure of inefficiency of the proposed algorithm, compared to the algorithm in [17] which reports exactly the nodes in \( V_C \) (although, by Proposition 2 DCD returns a set \( V_D \neq \emptyset \) if and only if there is at least a cycle in \( G_J \)).

According to the results, for \( n = 20 \) the number of messages...
In this paper we provide a methodology to detect and remove cycles in a directed graph. This is done by exploiting an underlying communication topology. The proposed algorithms have several advantages, with respect to the state of the art, in terms of completion time and bandwidth. Future work will be aimed at inspecting the effectiveness of the proposed approach when the agents communicate over communication topologies that do not have the same structure as the relational topology, or whose communication topology is not fixed. Moreover, we will assess the value of the maximum constant $c'$ for which the mSAS problem is APX-Hard, and we will study the performance and characteristics of the algorithms over different topologies.

A last envisaged future work direction is to investigate efficient distributed strategies to approximate the optimal swapping. A possible approach to do this is to resort to a Fundamental Cycle Basis [26], [27], identifying a set of fundamental cycles such that each other cycle can be expressed as a combination of them. The idea is to modify the fundamental cycles in order to obtain an acyclic graph.

**ACKNOWLEDGEMENT**

We would like to thank the anonymous reviewers for their valuable suggestions and insights.

**REFERENCES**


Gabriele Oliva received the Laurea degree and the Ph.D in Computer Science and Automation Engineering in 2008 and 2012, respectively, both at University Roma Tre of Rome, Italy. He is currently assistant professor in Automatic Control at the University Campus Bio-Medico of Rome, Italy. His main research interests include distributed systems, applications of graph theory in technological and biological systems, and Critical Infrastructure Protection.

Roberto Setola received the Laurea degree in Electronic Engineering (1992) and the PhD in Control Engineering (1996) from the University of Naples “Federico II”. From 1999 to 2004, he served at the Italian Prime Minister’s Office. He is currently Associate Professor of Automatic Control at the University CAMPUS Bio-Medico, where he is the head of the Complex System and Security Laboratory and the Director of the Post-graduate Master Program in Homeland Security. He was responsible for the Italian Government Working Group on Critical Infrastructure Protection (CIP) and a member of the GR Senior Information Experts’ group for CIP. He has been the coordinator of several EU projects and he authored seven books and more than 150 scientific papers. His main research interests are the simulation, modeling and control of complex systems and Critical Infrastructure Protection.

Luigi Glielmo was born in 1960; he holds a “laurea” degree in Electronic Engineering and a research doctorate in Automatic Control. He taught at University of Palermo, at University of Naples Federico II and then at University of Sannio at Benevento where he is now a professor of Automatic Control. His research interests over the years include singular perturbation methods, model predictive control methods, automotive controls, deep brain stimulation modeling and control, smart-grid control. He co-authored more than 130 papers on international archival journals or proceedings of international conferences, co-edited two books, and holds three patents. He seated in the editorial boards of prestigious archival journals, has been chair of the IEEE Control Systems Society Technical Committee on Automotive Controls. He has been head of the Department of Engineering of University of Sannio from 2001 to 2007 where he is currently Rector’s delegate for technology transfer and coordinator of the PhD course on Information Technologies for Engineering.

Christoforos N. Hadjicostis (M’99, SM’05) received the S.B. degrees in electrical engineering, computer science and engineering, and in mathematics, the M.Eng. degree in electrical engineering and computer science in 1995, and the Ph.D. degree in electrical engineering and computer science in 1999, all from Massachusetts Institute of Technology, Cambridge. In 1999, he joined the Faculty at the University of Illinois at Urbana-Champaign, where he served as Assistant and then Associate Professor with the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Information Trust Institute. Since 2007, he has been with the Department of Electrical and Computer Engineering, University of Cyprus, where he is currently Professor and Dean of Engineering. His research focuses on fault diagnosis and tolerance in distributed dynamic systems, error control coding, monitoring, diagnosis and control of large-scale discrete-event systems, and applications to network security, anomaly detection, energy distribution systems, medical diagnosis, biosequencing, and genetic regulatory models. He currently serves as Associate Editor of IEEE Transactions on Automatic Control, and IEEE Transactions on Automation Science and Engineering; he has also served as Associate Editor of IEEE Transactions on Control Systems Technology, and IEEE Transactions on Circuits and Systems I.