Evaluating Network Rigidity in Realistic Systems: Decentralization, Asynchronicity, and Parallelization

Ryan K. Williams, *Student Member, IEEE*, Andrea Gasparri, *Member, IEEE*, Attilio Priolo, and Gaurav S. Sukhatme, *Fellow, IEEE*

***Abstract*—In this paper, we consider the problem of evaluating the rigidity of a planar network, while satisfying common objectives of real-world systems: decentralization, asynchronicity, and parallelization. The implications that rigidity has in fundamental multi-robot problems, e.g., guaranteed formation stability and relative localizability, motivates this work. We propose the decentralization of the pebble game algorithm of Jacobs et. al., an $O(n^2)$ method that determines the generic rigidity of a planar network. Our decentralization is based on asynchronous messaging and distributed memory, coupled with auctions for electing leaders to arbitrate rigidity evaluation. Further, we provide a parallelization that takes inspiration from gossip algorithms to yield significantly reduced execution time and messaging. An analysis of the correctness, finite termination, and complexity is given, along with a simulated application in decentralized rigidity control. Finally, we provide Monte Carlo analysis in a *Contiki* networking environment, illustrating the real-world applicability of our methods, and yielding a bridge between rigidity theory and realistic interacting systems.**

***Index Terms*—Networked Robots; Distributed Robot Systems; Asynchronous and Parallel Communication; Graph Rigidity.**

I. Introduction

MULTI-ROBOT networks remain among the areas of interest at the forefront of robotics research, particularly given the steady advancement of wireless communication, embedded computation, and hardware platforms. Intuitively, networks of intelligently interacting robots provide significant advantages over the single-agent alternative; for example scalability, failure robustness, spatiotemporal efficiency, heterogeneity, etc. As recent work has demonstrated, multi-robot investigations are far-reaching across various disciplines, ranging from sampling, tracking, and coverage [1]–[3], mobility and topology control [4]–[6], to general agent agreement problems [7]–[9].

In modeling and analyzing multi-robot networks, research balances between accuracy in approximating realistic systems, and ease of technical analysis, most typically in understanding mobility, communication, and sensing. Here we take the former approach, considering a problem that underlies *fundamental* objectives in multi-robot research, while operating under the commonly desired parameters of real-world systems: *decentralized* implementation where information is exchanged only in local neighborhoods, *asynchronicity* in communication, and *parallelization* of agent actions to maximize efficiency. Our problem of interest in this work is the evaluation of the *rigidity* property of an interconnected system of intelligent agents, e.g., robots, sensors, etc. A relatively under-explored topic in the area of multi-agent systems, rigidity has important implications particularly for mission objectives requiring collaboration. For example, rigidity is vital for guaranteeing stability in controlling formations of mobile vehicles, when only relative inter-agent information is available [10]–[15]. Further, when a global frame of reference is inaccessible, rigidity becomes a necessary and under certain conditions sufficient condition for localization tasks with distance or bearing-only measurements [16]–[19]. Rigidity is also a necessary component of *global rigidity* [20]–[22], which can further strengthen the guarantees of formation stability and localizability. We point out that it is typical in the literature to *assume* rigidity properties of the network in order to achieve multi-agent behaviors, however few works provide means of evaluating or achieving network rigidity in a dynamic manner, or under the network conditions considered here.

The general study of rigidity has a rich history in various contexts of science, mathematics, and engineering [21]–[28]. In [27], combinatorial operations are defined which preserve rigidity, with works such as [10], [12] extending the ideas to multi-robot formations. In [29] an algorithm is proposed for generating rigid graphs in the plane based on the Henneberg construction [27], however from a centralized perspective. Similarly, [30] defines decentralized rigid constructions that are edge length optimal, however provide no means of determining an unknown graph's rigidity properties. The work [31] defines a rigidity eigenvalue for infinitesimal rigidity evaluation and control, however such efforts remain centralized and require continuous communication and computational resources.

As opposed to previous work, we propose a *decentralized* method of evaluating *generic* graph rigidity in the plane, *without* a priori topological information, to our knowledge the first such effort, particularly in a multi-agent context. To this end, we decentralize in an *asynchronous* manner the *pebble game* proposed by Jacobs and Hendrickson in [28], an algorithm that determines in $O(n^2)$ time the combinatorial rigidity of a network, and a spanning edge set defining the minimally rigid subcomponent of the graph. Specifically, we propose a leader election procedure based on distributed auctions that manages the sequential nature of the pebble game in

R. K. Williams and G. S. Sukhatme are with the Departments of Electrical Engineering and Computer Science at the University of Southern California, Los Angeles, CA 90089 USA (rkwillia@usc.edu; gaurav@usc.edu).

A. Gasparri and A. Priolo are with the Department of Engineering, University of Roma Tre, Via della Vasca Navale, 79. Roma, 00146, Italy (gasparri@dia.uniroma3.it; priolo@dia.uniroma3.it).

This work was partially supported by the ONR MURI program (award N00014-08-1-0693), the NSF CPS program (CNS-1035866), the NSF grant CNS-1213128, a fellowship to R. K. Williams from the USC Viterbi School of Engineering, and partially by the Italian grant FIRB "Futuro in Ricerca", project NECTAR, code RBFR08QWUV, funded by the Italian Ministry of Research and Education (MIUR).

a decentralized setting, together with a distributed memory architecture. Further, an asynchronous messaging scheme preserves local-only agent interaction, as well as robustness to delays, failures, etc. Towards network efficiency, we extend our decentralization by parallelizing a portion of the rigidity evaluation, taking inspiration from *gossip* messaging, yielding significant improvements in execution time and communication. To illustrate our contributions, we provide a thorough analysis of the correctness, finite termination, and complexity of our propositions, along with an illustration of decentralized rigidity control. Finally, we provide Monte Carlo analysis of our algorithms in a *Contiki* networking environment, illustrating the real-world applicability of our methods.

Although a few recent works have begun to investigate rigidity evaluation or control [29]–[32], they provide graph constructions or centralized control relying on expensive estimation techniques. We seek decentralization specifically to enhance scalability and robustness as network size increases, and to serve systems where centralized operation may be difficult or impossible. Our contributions therefore aim to bridge the gap between fundamentally important multi-agent behaviors and realistic rigidity evaluation in networked systems, ultimately moving towards robotic/sensor systems with achievable rigidity-based behaviors (as in our previous work [33]).

In summary, the major contributions of this paper are as follows:

- A leader-based, asynchronous decentralization of the classically centralized and serial pebble game algorithm, yielding a decentralized method for planar rigidity evaluation.
- A study of exploiting structural properties of rigidity for parallelization of our decentralized algorithm.
- A characterization of our algorithms in the real-world Contiki networking environment, with a full codebase release for use in the robotics community.

A preliminary portion of this paper appeared in [34], compared to which we provide expanded analysis and correctness proofs, a complete treatment of parallelization, expanded rigidity control simulations, and a Monte Carlo analysis in the Contiki environment demonstrating applicability under realistic conditions.

The outline of the paper is as follows. In Section II, we provide preliminary materials including agent and network models, a model of algorithm execution, and primers on rigidity theory and the pebble game. A decentralization of the pebble game is presented in Section III, with a parallelization given in Section IV. Simulation results are provided in Section V, with concluding remarks as well as directions for future work are stated in Section VI. Finally, technical algorithm details and related proofs are given in the Appendix.

II. Preliminaries and Formulation

A. Agent, Network, and Execution Models

Consider a system composed of n agents indexed by $\mathcal{I} = \{1, \dots, n\}$ operating in $\mathbb{R}^2$, each possessing computation and communication capabilities, denoting by (i, j) a bi-directional communication link between agents i and j. The agent may also be mobile, and thus we assume basic continuous dynamics

$$\dot{x}_i = f(\mathbf{x}) \tag{1}$$

where $x_i, f(x_i) \in \mathbb{R}^2$ are the position and the velocity control input for an agent $i \in \mathcal{I}$, respectively, and $\mathbf{x} \in \mathbb{R}^{2n}$ is the vector of stacked agent positions. For the purposes of integration into a motion control architecture (Section V-A) it is further assumed that each robot can sense other nearby robots and obstacles, yielding the displacement $d_{ij} \in \mathbb{R} \triangleq \|x_{ij}\| \triangleq \|x_i - x_j\|$.

To describe the interconnected system formally, we define *undirected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, having vertices $\mathcal{V} = \{v_1, \dots, v_n\}$ associated with each agent $i \in \mathcal{I}$, and edge set $\mathcal{E} = \{(i, j) \mid i, j \in \mathcal{V}\}$ with *unordered* pairs (i, j), where by definition $(i, j) \in \mathcal{E} \Leftrightarrow (j, i) \in \mathcal{E}, \forall\, i \neq j \in \mathcal{I}$, excluding the possibility for self loops, $(i, i) \notin \mathcal{E}, \forall\, i \in \mathcal{I}$. Agents i and j with an edge $(i, j) \in \mathcal{E}$ are referred to as *neighbors*, where the set of neighbors for the ith agent is given by $\mathcal{N}_i = \{v_j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$.

Assumption 1 (Connectedness): We assume the network topology $\mathcal{G}$ is *connected* for all time to guarantee all agents can participate in rigidity evaluation (Sections III and IV), that is for every pair of nodes i, j there exists a sequence of nodes in $\mathcal{G}$ that are adjacent and connect i, j. Notice that this assumption is trivially satisfied in rigid networks.

As our concern in this work is to operate under the typical parameters of realistic interacting systems, we assume an *asynchronous* model of time, where each agent $i \in \mathcal{I}$ has a clock which ticks according to some discrete distribution with finite support[1], independently of the clocks of the other agents [35], allowing also for the possibility of delayed communication over links $(i, j) \in \mathcal{E}$. Equivalently, this corresponds to a global clock having time-slots $[t_k, t_{k+1})$ which discretizes system time according to clock ticks, where for convenience we will use simply t to denote time [36]. Such assumptions induce asynchronicity in both agent computation and the broadcast and reception of inter-agent messages. First, we make the following assumptions concerning agent execution:

Assumption 2 (Agent execution): Each agent $i \in \mathcal{I}$ executes according to an algorithm on ticks of their clock, handling messages from neighbors $j \in \mathcal{N}_i$ and sending messages if dictated by the execution. *Local execution* is assumed to consist of atomic logic and message handling, that is all local algorithmic state, denoted $\mathbf{X}_i$, is assumed to be without race conditions due to asynchronicity.

A coordinated algorithm execution with associated stopping condition can then be defined as follows:

Definition 2.1 (Coordinated execution): A coordinated *algorithm execution* is given as a sequence of ticks t_k and therefore local execution and asynchronous messaging, yielding a *terminal state* upon some discrete network stopping condition

$$f_{stop} \in \{0, 1\} \triangleq f(\{\mathbf{X}_1, \dots, \mathbf{X}_n\}) \tag{2}$$

dependent on the execution states of the network agents. It is assumed that (2) can be computed using distributed techniques, e.g., consensus [37], as will be demonstrated in our proposed

[1]Notice that such an assumption allows us to appropriately characterize finite algorithm termination.

algorithms. After the stopping condition is observed the agents enter into an *idle* state where no execution occurs.

Finally, to guarantee soundness with respect to network communication and asynchronicity, we make the following assumptions:

Assumption 3 (Asynchronous messaging): We assume each agent $i \in \mathcal{I}$ treats messages received from neighbors $j \in \mathcal{N}_i$ in a *first-in-first-out* (FIFO) manner, guaranteeing soundness with respect to our proposed algorithm executions. Further, the possibility of communication failure is handled with *best-effort* messaging, i.e., there exists an underlying communication control layer where a best effort is made to deliver packets in the network. Specifically, we assume that the best effort guarantees message reception in finite time, or equivalently a message failure can be handled appropriately with respect to the algorithms that will be discussed.

B. Rigidity Theory

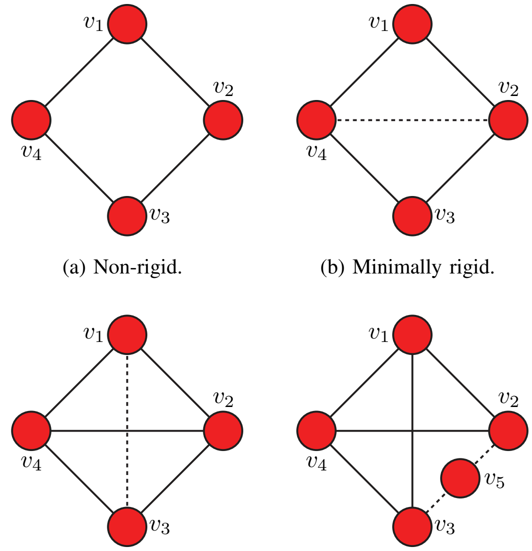
The primary concern of this work is the *rigidity* property of the underlying graph $\mathcal{G}$ describing the network topology, specifically as rigid graphs imply guarantees for example in both localizability and formation stability of multi-robot systems [12]. To begin, we recall the intuition of how rigidity is recognized in a planar graph, following the exposition of [28]. Clearly, graphs with many edges are more likely to be rigid than those with only a few, specifically as each edge acts to constrain the degrees of freedom of motion of the agents in the graph. In $\mathbb{R}^2$, there are $2n$ degrees of freedom in a network of n agents, and when we remove the three degrees associated with rigid translation and rotation, we arrive at $2n-3$ degrees of freedom we must constrain to achieve rigidity. Each edge in the graph can be seen as constraining these degrees of freedom, and thus we expect $2n-3$ edges will be required to guarantee a rigid graph. In particular, if a subgraph containing k vertices happens to contain more than $2k-3$ edges, then these edges cannot all be required for constraining the degrees of motion, i.e., they cannot be all *independent*. Our goal in evaluating rigidity is thus to identify the $2n-3$ edges that independently constrain the motion of our agents, precisely describing a networks underlying rigid component.

We now provide a brief technical overview of the above intuition, and direct the reader to [23]–[27] for an in depth review of rigidity theory. First, we require the notion of a graph embedding in the plane, captured by the *framework* $\mathbb{F}_p \triangleq (\mathcal{G}, p)$ comprising graph $\mathcal{G}$ together with a mapping $p\colon \mathcal{V} \to \mathbb{R}^2$, assigning to each node in $\mathcal{G}$, a location in $\mathbb{R}^2$. The natural embedding for us is to assign each node the position x_i associated with each agent, defined by the mapping $p(i) = x_i$, otherwise known as a *realization* of $\mathcal{G}$ in $\mathbb{R}^m$. Therefore, a framework describes both the communication topology of a multi-agent system, and the spatial configuration of each agent in the plane.

The *infinitesimal motion* of $\mathbb{F}_p$ can be described by assigning to the vertices of $\mathcal{G}$, a velocity $\dot{p}_i \triangleq \dot{x}_i \in \mathbb{R}^2$ such that

$$(\dot{x}_i - \dot{x}_j) \cdot (x_i - x_j) = 0, \qquad \forall\, (i,j) \in \mathcal{E} \tag{3}$$

where $\cdot$ is the standard dot product over $\mathbb{R}^m$. That is, edge lengths are preserved, implying that no edge is compressed or



(a) Non-rigid. (b) Minimally rigid.

(c) Non-minimally rigid. (d) Non-infinitesimally rigid.

Fig. 1. Example graphs demonstrating several embodiments of rigidity, where dashed links indicate edges that have been added to form new networks. Notice that all links in graphs (a), (b), and (d) are independent, while edge (v_1, v_3) in (c) is redundant.

stretched over time. The framework is said to undergo a *finite flexing* if p_i is differentiable and edge lengths are preserved, with *trivial flexings* defined as translations and rotations of $\mathbb{R}^2$ itself. If for $\mathbb{F}_p$ all infinitesimal motions are trivial flexings, then $\mathbb{F}_p$ is said to be *infinitesimally rigid*. Otherwise, the framework is called *infinitesimally flexible*, as in Fig. 1a, where v_1 and v_3 can move inward with v_2 and v_4 moving outward, while preserving edge lengths [27]. In the context of a robotic network, rigid infinitesimal motion corresponds to movement of the ensemble in which the distances between robots remain fixed over time.

The infinitesimal rigidity of $\mathbb{F}_p$ is tied to the specific embedding of $\mathcal{G}$ in $\mathbb{R}^2$, however it has been shown that the notion of rigidity is a *generic* property of $\mathcal{G}$, specifically as *almost all* realizations of a graph are either infinitesimally rigid or flexible, i.e., they form a dense open set in $\mathbb{R}^2$ [38]. Thus, we can treat rigidity from the perspective of $\mathcal{G}$, abstracting away the necessity to check every possible realization. The first such *combinatorial* characterization of graph rigidity was described by Laman in [23], and is summarized as follows[2]:

Theorem 2.1 (Graph rigidity, [23]): A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with realizations in $\mathbb{R}^2$ having $n \geq 2$ nodes is rigid if and only if there exists a subset $\bar{\mathcal{E}} \subseteq \mathcal{E}$ consisting of $|\bar{\mathcal{E}}| = 2n-3$ edges satisfying the property that for any non-empty subset $\hat{\mathcal{E}} \subseteq \bar{\mathcal{E}}$, we have $|\hat{\mathcal{E}}| \leq 2k-3$, where k is the number of nodes in $\mathcal{V}$ that are endpoints of $(i,j) \in \hat{\mathcal{E}}$.

Laman's notion of graph rigidity is also referred to as *generic rigidity*, and is characterized by the *Laman conditions* on the network's subgraphs. Intuitively, the concept of rigidity can be thought of in a physical way, that is if the graph were a bar and joint framework, it would be mechanically rigid against

[2]The extension of Laman's conditions to higher dimensions is at present an unresolved problem in rigidity theory.

external and internal forces. However, we point out that the rigidity of an underlying graph is purely a topological property. A network of agents that is described by a rigid graph is not necessarily mechanically rigid. Instead, the structure of its interconnections, in our case robot-to-robot communication, possesses the combinatorial properties of the above Laman conditions.

Denote by $\mathcal{G}_{\mathbb{R}}$ the set of all rigid graphs in $\mathbb{R}^2$, and the graph $\mathcal{S} = (\mathcal{V}, \bar{\mathcal{E}})$ satisfying Theorem 2.1 a *Laman subgraph* of $\mathcal{G}$. It follows from Theorem 2.1 that any rigid graph in the plane must then have $|\mathcal{E}| \geq 2n - 3$ edges, with equality for *minimally rigid* graphs. The impact of each edge on the rigidity of $\mathcal{G}$ is captured in the notion of *edge independence*, a direct consequence of Theorem 2.1:

Definition 2.2 (Edge independence, [28]): Edges $(i,j) \in \mathcal{E}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are *independent* in $\mathbb{R}^2$ if and only if no subgraph $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ has $|\bar{\mathcal{E}}| > 2|\bar{\mathcal{V}}| - 3$. A set of independent edges will be denoted by $\mathcal{E}^*$, while the graph over $\mathcal{E}^*$ is denoted by $\mathcal{G}^*$.

The above conditions imply that a graph is rigid in $\mathbb{R}^2$ if and only if it possesses $|\mathcal{E}^*| = 2n - 3$ independent edges, where edges that do not meet the conditions of Definition 2.2 are called *redundant*. Thus, in determining the rigidity of $\mathcal{G}$, we must verify the Laman conditions to discover a suitable set of independent edges $\mathcal{E}^*$. We refer the reader Fig. 1 for a depiction of graph rigidity. Notice that the graph in Fig. 1a is non-rigid as it does not fulfill the basic $2n-3$ edge condition of Laman. In adding edge (v_2, v_4) we then generate the minimally rigid graph of Fig. 1b as every subgraph of k vertices has at most $2k - 3$ edges. Further addition of (v_1, v_3) yields the non-minimally rigid graph in Fig. 1b, precisely as the graph possesses greater than $2n-3$ edges. Finally, Fig. 1d represents a non-infinitesimally rigid graph as there exist non-trivial motions that preserve edge lengths, i.e., v_5 moves independently while the remaining nodes rotate together, but it also is a generically rigid graph as the underlying Laman conditions are satisfied.

C. A Pebble Game for Evaluating Generic Rigidity

To lessen the exponential complexity of the Laman conditions we consider the *pebble game* proposed by Jacobs and Hendrickson in [28]. A brief overview of the centralized pebble game will be given here, beginning with a useful characterization of the Laman conditions and edge independence:

Theorem 2.2 (Laman restated, [28]): For graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the following statements are equivalent:

- All $(i,j) \in \mathcal{E}$ are independent in $\mathbb{R}^2$.
- For each $(i,j) \in \mathcal{E}$, the graph formed by *quadrupling* (i,j), i.e., adding 4 *virtual* copies of (i,j) to $\mathcal{E}$, has no subgraph $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ in which $|\bar{\mathcal{E}}| > 2|\bar{\mathcal{V}}|$.

Theorem 2.2 represents the Laman condition, i.e., a subgraph of k vertices can possess at most $2k - 3$ edges, through the simple quadrupling operation. In other words, if we add 3 virtual copies of an edge to some subgraph, and the condition $|\bar{\mathcal{E}}| \leq 2k$ is met in this subgraph, it must be the case that the property $|\bar{\mathcal{E}}| \leq 2k - 3$ holds in the original subgraph. This intuition can be further extended to *incrementally* evaluate edge independence:

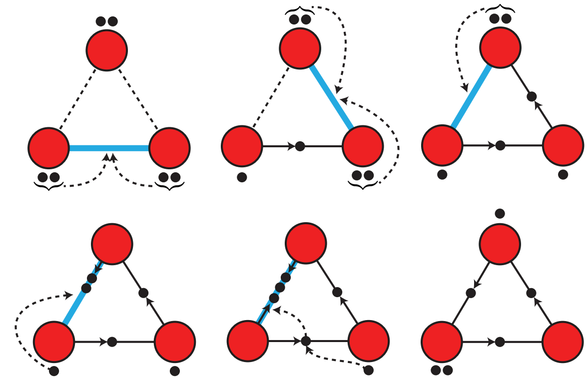


Fig. 2. An example of the pebble game for a rigid graph with $n = 3$ with progression from left to right. Pebbles are given by black dots, quadrupled edges by thick links (blue), pebble shifts by dashed arrows, and the local assignment of pebbles by black arrows. Graph edges that remain to be quadrupled are dashed. We have here $|\mathcal{E}^*| = 3$.

Lemma 2.1 (Edge quadrupling, [28]): Given an independent edge set $\mathcal{E}^*$, an edge $(i,j) \notin \mathcal{E}^*$ is independent of $\mathcal{E}^*$ if and only if the graph formed by the union of $\mathcal{E}^*$ and quadrupled edge (i,j) has no subgraph $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ in which $|\bar{\mathcal{E}}| > 2|\bar{\mathcal{V}}|$.

The above Lemma provides us with a simple process for testing rigidity: we incrementally quadruple edges in the graph, check the induced subgraph property, and continue until we have either discovered $2n - 3$ independent edges or we have exhausted $\mathcal{E}$. However, this process alone does not save us from the exponential complexity of verifying the subgraph property. To this end, [28] provides a natural simplification in the following *pebble game*:

Definition 2.3 (The pebble game, [28]): Considering a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where we associate an agent with each $v \in \mathcal{V}$, give to each agent two pebbles which can be assigned to an edge in $\mathcal{E}$. Our goal in the pebble game is to assign the pebbles in $\mathcal{G}$ such that all edges are covered, i.e., a *pebble covering*. In finding a pebble covering, we allow the assignment of pebbles by agent i *only* to edges incident to v_i in $\mathcal{G}$. Further we allow pebbles to be rearranged only by removing pebbles from edges which have an adjacent vertex with a free pebble, such that the free pebble is shifted to the assigned pebble, freeing the assigned pebble for assignment elsewhere. Thus, if we consider pebble assignments as directed edges exiting from an assigning agent i, when a pebble is needed in the network to cover an edge (i,j), a *pebble search* over a directed network occurs. If a free pebble is found, the rules for local assignment and rearranging then dictate the pebble's return and assignment to (i,j).

Lemma 2.2 (Pebble covering, [28]): In the context of the pebble game of Definition 2.3, if there exists a pebble covering for an independent edge set $\mathcal{E}^*$ with a quadrupled edge $(i,j) \notin \mathcal{E}^*$, there is no subgraph violating the conditions of Lemma 2.1, and the set $\mathcal{E}^* \cup (i,j)$ is independent.

Rigidity evaluation now operates as follows: every edge $e \in \mathcal{E}$ is quadrupled, and an attempt to expand the current pebble covering for $\mathcal{E}^*$ to each copy of e is made, with success resulting in $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup e$ and termination coming when $|\mathcal{E}^*| = 2n - 3$. Intuitively, an agent's pebbles represent its possible commitments to the network's subgraphs, while maintaining the subgraph conditions of Lemma 2.1, or in a physical way the degrees of freedom of motion in $\mathbb{R}^2$. Further, the edge quadrupling operation and the pebble game effectively cast the Laman conditions on subgraphs in terms of a matching problem. That is, as each agent is given 2 pebbles, and each of 4 instances of a considered edge must be assigned a pebble, we implicitly verify the $2k-3$ edge condition of Laman when these 4 pebbles are found in a subgraph containing k vertices. This is the case precisely because each previously considered edge is assigned a single pebble.

The *centralized* pebble game of Jacobs is depicted in Algorithm 1, with an illustration of the quadrupling and pebble search procedure depicted in Fig. 2. In the simple three node graph shown, there are six available pebbles that can only be assigned locally. Thus, in quadrupling each edge and finding four pebbles, the subgraph conditions of Laman are incrementally verified. The progression is given from left to right and clockwise in the figure, with pebbles given by black dots, quadrupled edges by thick links (blue), pebble shifts by dashed arrows, and the local assignment of pebbles by black arrows. Graph edges that remain to be quadrupled are dashed. Notice that in discovering the pebble to cover the final copy of the last quadrupled edge (bottom middle pane), a search occurs over the directed graph formed by previous pebble assignments.

Algorithm 1 The centralized pebble game [28].

procedure PEBBLEGAME($\mathcal{G} = (\mathcal{V}, \mathcal{E})$)
    Assign each v_i two pebbles, $\forall\, i \in \mathcal{I}$
    $\mathcal{E}^* \leftarrow \emptyset$
    for all $(i,j) \in \mathcal{E}$ **do**
        Quadruple (i,j) over $\mathcal{G}$
        Search for 4 pebbles, originating from v_i and v_j
        if *found* **then**
            Rearrange pebbles to cover quadrupled (i,j)
            $\triangleright$ Expand independent set, check rigidity:
            $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup (i,j)$
            if $|\mathcal{E}^*| = 2|\mathcal{V}| - 3$ **then**
                return $\mathcal{E}^*$
            end if
        end if
    end for
end procedure

III. AN ASYNCHRONOUS DECENTRALIZED PEBBLE GAME

The primary considerations in decentralizing the pebble game of [28] lie in the sequential building of the independent edge set $\mathcal{E}^*$, the storage of $\mathcal{E}^*$ and associated pebble assignments over a distributed network, and the search and rearranging of pebbles throughout the network. To deal with these issues, we summarize the high level components of our decentralization:

- **Leader election**: to control the sequential building of $\mathcal{E}^*$, lead agents are elected through auctions to examine their incident edges for independence. In determining edge independence, pebbles are *queried* from the network through inter-agent messaging in order to cover each copy of a quadrupled incident edge. Leadership then transfers to the next auction winner when the current leader's neighborhood has been exhausted.
- **Distributed storage**: independent edges and pebble assignments are localized to each agent, effectively distributing network storage. We then rely on messaging and proper agent logic to support pebble searches and shifts.
- **Local messaging**: as opposed to searching a centralized graph object for pebbles to establish edge independence, we endow the network with a pebble request/response messaging protocol to facilitate pebble searches.

Intuitively, our leader-based decentralization is an incremental rooting of pebble searches at appropriately elected network leaders, effectively partitioning rigidity evaluation as in Fig. 3. For convenience we will denote by $\mathbb{S}$ our decentralization of the serial pebble game of Algorithm 1.

In describing our algorithm we associate with each agent $i \in \mathcal{I}$ the following variables, with initialization indicated by $\leftarrow$:

- $\mathcal{P}_i \leftarrow \emptyset$: Pebble assignment set containing at most two edges $\{(i,j) \in \mathcal{E} \,|\, j \in \mathcal{N}_i\}$, that is incident edges (i,j) to which a pebble is associated. For convenience, we let $p_i = 2 - |\mathcal{P}_i| \in \{0,1,2\}$ denote agent i's free pebble count.
- $\mathcal{E}_i^* \leftarrow \emptyset$: Local independent edge set, containing edges $\{(i,j) \in \mathcal{E} \,|\, j \in \mathcal{N}_i\}$ for which quadrupling and pebble covering succeeds. By construction $\mathcal{E}^* = \bigcup_i \mathcal{E}_i^*$.

A. Leader Election

An execution of the $\mathbb{S}$ algorithm begins when an agent detects network conditions that require rigidity evaluation, e.g., verifying link deletion to preserve rigidity. The initiating agent begins by triggering an *auction* for electing an agent in the network to become the *leader*. Specifically, to each agent $i \in \mathcal{I}$ we associate a *bid* for leadership $r_i = [i, b_i]$ with $b_i \in \mathbb{R}_{\geq 0}$ indicating the agent's *fitness* in becoming the new leader, with $b_i = 0$ if agent i has previously been a leader, and $b_i \in \mathbb{R}_+$ otherwise. Denoting the local bid set by $\mathcal{R}_i = \{r_j \,|\, j \in \mathcal{N}_i \cup \{i\}\}$, the auction then operates according the following agreement process:

$$r_i(t^+) = \underset{r_j \in \mathcal{R}_i}{\operatorname{argmax}}(b_j) \tag{4}$$

where the notation t^+ indicates a transition in r_i after all neighboring bids have been collected through messaging. As $\mathcal{G}$ is assumed connected for all time, (4) converges *uniformly* to the largest leadership bid

$$r_i = \underset{r_j(0)}{\operatorname{argmax}}(b_j(0)), \quad \forall\, i,j \in \mathcal{I} \tag{5}$$

after some finite time [39], [40]. After convergence of (4) the winning agent then takes on the leadership role, with the previous leader relinquishing its status. The proposed auction mechanism allows us to decentralize the pebble game by assigning to each leader the responsibility of expanding $\mathcal{E}_i^*$ by evaluating only their incident edges for independence. Also, notice that previous leaders are never reelected due to $b_i = 0$

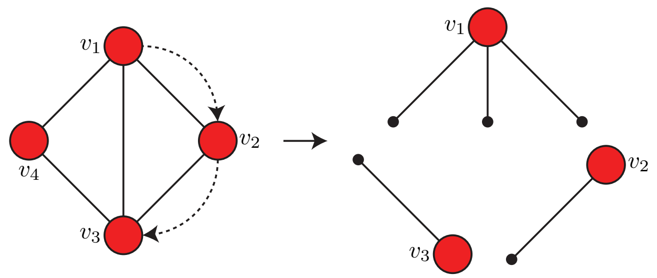


Fig. 3. Illustration of our leader-based decentralization concept for $n = 4$ agents. Each leader v_i inspects only the unevaluated portion of their neighborhood $\mathcal{N}_i$, passing leadership to a new agent (dashed arrow) through a distributed auction. Each leader applies messaging to establish the independence of their incident edges with respect to those of the previous leaders.

for such agents, and that the condition $b_i = 0, \forall\, i \in \mathcal{I}$ allows termination of the algorithm.

One would expect that as each leader expands the independent set sequentially that the *order* of election is meaningful. We characterize that relationship in the following:

Proposition 3.1 (Initial leader edges): All incident edges $\{(i, j) \in \mathcal{E} \,|\, j \in \mathcal{N}_i\}$ belonging to an initial leader i are members of the independent set $(i, j) \in \mathcal{E}^*$.

Proof: For each edge, a new node $j \neq i$ must be considered as no two edges of i can have the same endpoint and $\mathcal{E}^*$ is empty due to i being the initial leader. Therefore, every subgraph containing the edges and nodes incident to i must have $|\mathcal{E}_s| \leq 2|\mathcal{V}_s| - 3$ edges, where $\mathcal{V}_s$ are the nodes of the considered subgraph and $|\mathcal{E}_s| = |\mathcal{V}_s| - 1$ due to the subgraph's implicit tree structure. Thus, as there exists no subgraph violating Definition 2.2, the result follows. ■

As the agent with the largest bid is elected, the bids dictate the *order* of elected leaders and thus the edges that constitute the identified rigid subgraph. In other words, the bids can be applied based on the application. For example, if we assume each edge is assigned a weight which indicates its value in sensing or information, we could choose leader bids that are the sum of incident edge value. Then the resulting rigid subgraph would possess those edges that both establish rigidity and are the most valuable in the given application. Bids could also be chosen to reflect agent availability, processing capability, or the cardinality of incident edges, or they can be leveraged in terms of metrics related to mission objectives. The proposed auction technique therefore affords us control over $\mathcal{E}^*$ that goes beyond simply discovering the network's rigidity property.

B. Leader Tasks

After election, the primary task of the leader i is to continue the expansion of $\mathcal{E}^*$ by evaluating the independence of each edge $(i, j) \in \mathcal{E}_i \triangleq \{\mathcal{N}_i \,|\, \neg \mathit{beenLeader}(j)\}$, i.e., the set of unevaluated incident edges. In initializing $\mathcal{E}_i$ in such a way, incident edges (i, j) are considered only when the neighbor $j \in \mathcal{N}_i$ has not yet been a leader, as edges incident to a previous leader j have already been checked. This guarantees that network edges are considered only *once* for quadrupling and pebble covering. Also, note that each leader receives the current size of the independent edge set $|\mathcal{E}^*(t)|$ in initialization, by embedding $|\mathcal{E}^*(t)|$ in the leadership auction. This allows a leader to terminate the algorithm when $2n - 3$ independent edges have been identified.

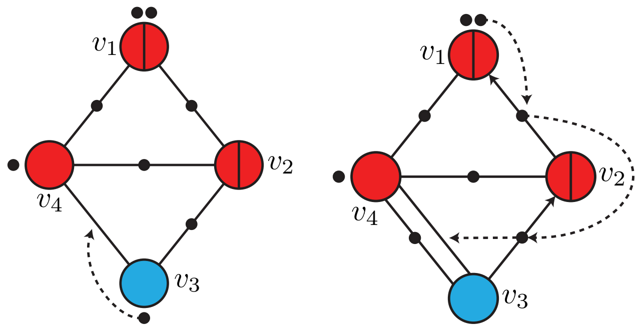
The leader executes the procedure LEADERRUN depicted in Algorithm 2, given in the Appendix, to accomplish the task of evaluating its incident edges. First, recall that in checking independence a pebble covering for each quadrupled edge $e_i \in \mathcal{E}_i$ must be determined. As the pebble information is distributed across the network, the lead agent must therefore *request* pebbles through messaging in an attempt to assign pebbles to e_i. After making a pebble request, the lead agent then pauses execution and waits for pebble *responses* before continuing; a method often referred to as *blocking*.

When there exists no unfulfilled pebble requests, the lead agent starts or resumes the quadrupling procedure on the current incident edge $e_i \in \mathcal{E}_i$, lines 3–11. For each copy of e_i, the leader searches for a pebble to cover e_i, first by looking locally for free pebbles, assigning e_i to $\mathcal{P}_i$ if found. If no local pebbles are available, the agent then sends a PEBBLEREQUESTMSG to the endpoint of the first edge to which a pebble is assigned, requesting a free pebble. If a pebble is received from this request, the quadrupling process continues, otherwise another request is sent to the endpoint of the second edge to which a pebble is assigned. In sending requests only along $(i, j) \in \mathcal{P}_i$, we properly evaluate independence with respect to $\mathcal{E}^*$, as each $(i, j) \in \mathcal{E}^*$ must have an assigned pebble from previous evaluations of independence.

As established by Lemma 2.2, the outcome of the quadrupling process, i.e., the existence of 4 free pebbles in the network, dictates the independence of edge e_i. If the leader fails to receive 4 pebbles to cover e_i, the edge is deemed redundant and evaluation moves to the next member of $\mathcal{E}_i$. On the other hand, for any edge e_i with a pebble covering, obtained through a combination of local assignment and pebble responses, the following actions are taken, lines 13–24. First, we return 3 pebbles to the endpoints of e_i leaving a single pebble on e_i to establish independence, and then add e_i to $\mathcal{E}_i^*$. If in adding e_i, $2n - 3$ independent edges have been identified, the leader sends a simple message to the network indicating that the graph is rigid, and the algorithm terminates. Otherwise, the leader moves to the next member of $\mathcal{E}_i$ and begins a new quadrupling process. When all members of $\mathcal{E}_i$ have been evaluated, the leader initiates the auction (4) to elect the next leader. The process of edge quadrupling, pebble requests, edge covering, and expansion of the independent set then continues from leader to leader until either the network is found to be rigid, or every agent has been a leader, indicating non-rigidity.

C. Inter-Agent Messaging

As each leader attempts to expand $\mathcal{E}_i^*$ through quadrupling each of its members, free pebbles are needed to establish a pebble covering. We facilitate the pebble search by defining asynchronous message PEBBLEREQUESTMSG, accompanied by response messages PEBBLEFOUNDMSG and PEBBLENOTFOUNDMSG, indicating the existence of free pebbles. The arrival of these messages then triggers message handlers that form the foundation of the pebble search mecha-



(a) First edge of quadrupling. (b) Second edge of quadrupling.

Fig. 4. Illustration of the first two pebble covering attempts for a quadrupling on edge $(3, 4)$. Agents v_1 and v_2 have previously been a leader, while agent v_3 (blue) is the current leader. Pebbles are depicted by solid black dots, requests by inter-agent arrows, and responses and shifts by dashed arrows.

nism. For technical details of the protocol, see the pseudocode given in the Appendix.

The reception of a PEBBLEREQUESTMSG initiates the handler HANDLEPEBBLEREQUEST depicted in Algorithm 3. Each pebble request is marked with a unique identifier, originating from the lead agent, defining the pebble search to which the request is a member and ensuring proper message flow in the network, lines 2–5. For unique requests, the receiving agent first attempts to assign local pebbles to the edge connecting the pebble requester, i.e., a pebble shift operation, lines 7–9. If a free pebble is available for the shift, a PEBBLEFOUNDMSG is sent in response, allowing the requester to free an assigned pebble for either local assignment or to itself respond to a pebble request. If instead the request recipient has no free pebbles, the agent forwards the request to the endpoints of its assigned pebbles, recording the original pebble requester such that responses can be properly returned, lines 11–12. Notice that this messaging logic not only facilitates the pebble shift and assignment rules of the original pebble game, but also eliminates the need for explicit message routing. Instead, it is previous pebble assignments that dictate message routing.

When the PEBBLEFOUNDMSG response to a pebble request is received it triggers the handler HANDLEPEBBLEFOUND depicted in Algorithm 4. Similar to the shifting action of HANDLEPEBBLEREQUEST, the agent first frees the local pebble assigned to the edge connecting the responder, line 2, and then uses the newly freed pebble depending on leader status. If the agent is currently the leader, line 4, the freed pebble is assigned locally to e_i, continuing the edge quadrupling process and relieving the request blocking condition. For non-lead agents, a pebble shift is performed to again free a pebble for a requesting agent, indicating the shift by returning a PEBBLEFOUNDMSG to the requester, lines 6–7.

Finally, the PEBBLENOTFOUNDMSG response to a pebble request initiates the handler HANDLEPEBBLENOTFOUND depicted in Algorithm 5. For both leaders and non-leaders, the lack of a free pebble initiates a further search in the network, along untraversed incident edges to which a pebble is assigned, line 3. However, if both available search paths have been exhausted, the leadership status of the receiver dictates the action taken. In the case of a non-leader, line 11, the response is simply returned to the original requester in order to initiate further search rooted from the requester. For a leader, lines 6–9, the lack of free pebbles in the network indicates precisely that the conditions of Lemma 2.1 do not hold, implying the currently considered edge e_i is *redundant*. The edge e_i is removed from consideration by returning all pebbles assigned during the covering attempt to the endpoints of e_i, and the process is moved to the next incident edge. A basic illustration of a snapshot of the $\mathbb{S}$ algorithm is given in Fig. 4.

D. Complexity Analysis

The complexity of $\mathbb{S}$ is promising for realistic decentralized operation:

Proposition 3.2 ($\mathbb{S}$ complexity): By construction, executions of the $\mathbb{S}$ algorithm have *worst-case* $O(n^2)$ messaging complexity and $O(n)$ storage scaling.

Proof: As the pebble game exhibits $O(n^2)$ complexity [28], our pebble messaging scales like $O(n^2)$. In applying leader auction (4) we incur $O(n^2)$ as we expend $O(n)$ auction messaging for $O(n)$ leaders. Equivalently, the centralized execution takes $O(n^2)$ and we simply apply an $O(n^2)$ decentralization to provide the algorithm with the appropriate runtime information. Thus, our overall algorithm will run with $O(n^2)$ complexity. Finally, the per-agent storage complexity scales like $O(n)$, the maximal cardinality of $\mathcal{N}_i$, as assignments to $\mathcal{E}_i$ occur over only edges incident to i, line 23 Algorithm 2. ■

The above result demonstrates that our proposed $\mathbb{S}$ algorithm represents a fully decentralized and efficient solution to the planar generic rigidity evaluation problem, providing the opportunity to exploit the vast advantages of network rigidity in realistic robotic networks. Technical analysis and detailed pseudocode for the $\mathbb{S}$ algorithm can be found in the Appendix.

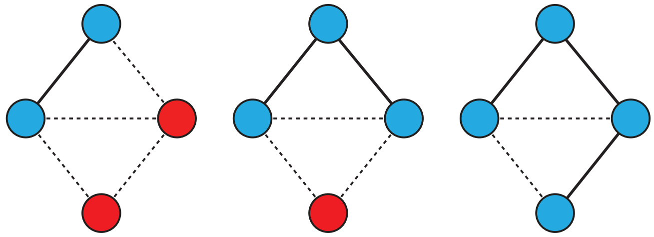
IV. EXPLOITING STRUCTURE TOWARDS PARALLELIZATION

To fully exploit a distributed multi-agent system, we seek a parallelization of the algorithm proposed in Section III, with the goal of reducing the overall execution time of rigidity evaluation, and rendering real-world application feasible. It turns out that evaluating network rigidity is intrinsically serial and centralized in nature, making it difficult to *asymptotically* reduce the computational complexity through parallelization. Instead, we aim to provide a parallelization that is advantageous under realistic circumstances, yielding both non-trivial runtime improvements and uses for building rigid networks, with no additional hardware or communication requirements. At a high level, our scheme consists of identifying local edge addition operations that preserve independence, and allowing the agents to apply these rules simultaneously to build a set of independent edges. We will develop these ideas in the sequel and direct the reader to Remark 4.2 for a complete summary of the advantages of parallelization.

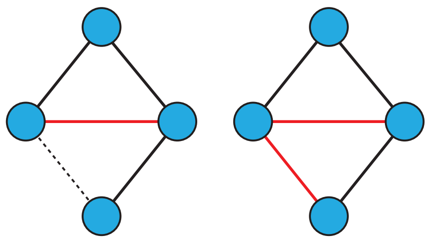
A. Independence Preserving Operations

Let us begin by formally defining addition and subtraction operations for graph edges as follows.

Definition 4.1 (Edge addition/subtraction [10]): Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and let the graph augmented with an



(a) The endpoint expansion rule (Definition 4.3).



(b) The two incident edge rule (Definition 4.4).

Fig. 5. Illustration of a sequence of the endpoint expansion rule (a) and the two incident edge rule (b). Edge addition operations to $\mathcal{E}^*(k)$ are given by red links, endpoint membership in $\mathcal{G}^*(k+1)$ is depicted by blue nodes, and $\mathcal{E}^*(k)$ is shown by black links. Notice in the illustrated minimally rigid graph, a combination of EER and TIER operations identifies fully the independent edge set $\mathcal{E}^*$.

edge e be denoted $\mathcal{G}^+ = (\mathcal{V}, \mathcal{E} \cup \{e\})$. Similarly, the graph $\mathcal{G}$ with e removed is denoted by $\mathcal{G}^- = (\mathcal{V}, \mathcal{E} \setminus \{e\})$. We refer to the operation $[\cdot]_e^+$ such that $\mathcal{G}^+ = [\mathcal{G}]_e^+$ as *edge addition*. Likewise, the operation $[\cdot]_e^-$ such that $\mathcal{G}^- = [\mathcal{G}]_e^-$ as *edge subtraction* (or deletion).

Now we are prepared to consider *independence preserving* graph operations. Specifically:

Definition 4.2 (Independence preserving operations): We call the edge operations $[\cdot]_e^+$ and $[\cdot]_e^-$ over graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ having independent edges $\mathcal{E}$ satisfying Definition 2.2 *independence preserving* (IP) if $\mathcal{E} \cup \{e\}$ and $\mathcal{E} \setminus \{e\}$ are themselves independent, respectively. Clearly, all operations $[\cdot]_e^-$ over independent edges $\mathcal{E}$ are independence preserving. Also, we have that addition operations $[\cdot]_e^+$ that are *not* independence preserving imply e is redundant with respect to $\mathcal{E}$.

Thus, we seek IP edge addition operations that enable the construction of $\mathcal{E}^*$ in a parallel fashion. Then, given an initial independent set $\mathcal{E}^*(0) = \emptyset$ with associated graph $\mathcal{G}^* = (\mathcal{V}, \mathcal{E}^*)$, we can generate the sequence

$$\mathcal{G}^*(0) = (\mathcal{V}, \emptyset) \quad \mathcal{G}(k) = [\mathcal{G}(k-1)]_e^+, \; k = 1, \ldots, m \tag{6}$$

where if each $[\cdot]_e^+$ is independence preserving, the resulting graph $\mathcal{G}^*(m)$ possesses independent edges. Then, if each operation $[\cdot]_e^+$ is local to the endpoints of edge e, sequence (6) can be achieved in parallel.

To identify such operations, we take inspiration from the *Henneberg construction*, a sequence of node and edge additions that iteratively builds a minimally rigid graph [27]. First, consider a simple rule based on the membership of v_i or v_j as endpoints in $\mathcal{G}^*(k)$:

Definition 4.3 (Endpoint expansion rule): Consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the associated node set $\widehat{\mathcal{V}} = \{v_i \in \mathcal{V} \mid \exists \; j \in \mathcal{I}, (i,j) \in \mathcal{E}\}$, containing the nodes in $\mathcal{V}$ which are endpoints of edges in $\mathcal{E}$. Define the *endpoint expansion rule* (EER) as the edge addition operation $[\mathcal{G}]_e^+$ possessing the property that $|[\widehat{\mathcal{V}}]_e^+| > |\widehat{\mathcal{V}}|$, where $[\widehat{\mathcal{V}}]_e^+$ are the endpoints of $\mathcal{G}^+$. Notice that for an EER operation it trivially holds that $2 \geq |[\widehat{\mathcal{V}}]_e^+| - |\widehat{\mathcal{V}}| \geq 1$.

Clearly the endpoint expansion rule 4.3 is limiting in terms of the identified set $\mathcal{E}^*$, specifically as the identified set can be described as the union of spanning trees over $\mathcal{G}$, a direct consequence of expanding endpoints in a graph, as in Fig. 5a. Thus, we can further consider a two edge rule that is also independence preserving:

Definition 4.4 (Two incident edge rule): Consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the augmented graph $\mathcal{G}^+$ through addition of edge $e \triangleq (i,j)$. The *two incident edge rule* (TIER) is an edge addition operation $[\mathcal{G}]_e^+$ where it holds that $(\mathcal{N}_i^+ \leq 2) \vee (\mathcal{N}_j^+ \leq 2)$ over $\mathcal{G}^+$, with $(\mathcal{N}_i \geq 1) \wedge (\mathcal{N}_j \geq 1)$ over $\mathcal{G}$, otherwise the edge operation would constitute an endpoint expansion.

We illustrate the rules of Definitions 4.3 and 4.4 in Figs. 5a and 5b, respectively, with the independence preservation of the proposed rules given by Proposition A.4 in the Appendix. The EER and TIER operations indicate first that an independent set could be built incrementally as in (6), much like the original pebble game. However, instead of requiring inherently global pebble searches, the EER and TIER operations are distinctly local in nature, making them amenable to parallel implementation.

B. Gossip-like Messaging for Parallelization

We now take inspiration from the randomized communication scheme typical of *gossip* algorithms, e.g., [41], [42], to define the execution and messaging structure for partial parallelization of rigidity evaluation. Each agent i exchanges inter-neighbor messages in an attempt to assign incident edges $(i,j), \forall j \in \mathcal{N}_i$ to $\mathcal{E}_i^*$ according to the EER and TIER rules. Such a construction, denoted as algorithm $\mathbb{P}$, allows the network to determine a subset of independent edges $\mathcal{E}_\mathbb{P}^* \subseteq \mathcal{E}^*$ with significantly reduced execution time and messaging, as will be verified in Section V-B. The technical pseudocode for our parallelization is given in the Appendix.

To characterize the EER and TIER operations on the edges assigned to an agent's independent set, we associate with each agent i the variable *committed*$(i) \in \mathbb{Z}_{\geq 0}$. This *commitment* variable stores the cardinality of i as an endpoint of edges in the distributed independent edge set, or more formally, the node degree of v_i in the graph $\mathcal{G}^* = (\mathcal{V}, \cup_i \mathcal{E}_i^*)$. As the EER and TIER rules are effectively conditions on node degrees, as one would expect given the nature of the Laman conditions, the commitment variables guarantee that all edges added to the independent set are in fact independent.

As opposed to the leader-based execution of algorithm $\mathbb{S}$, here every agent executes concurrently according to Algorithm 6, with initialization

$$\mathcal{E}_i(0) \leftarrow \mathcal{N}_i, \quad e_i \nleftarrow (i,j) \in \mathcal{E}_i(0) \quad \forall\, i \in \mathcal{I} \tag{7}$$

where we use notation $\nleftarrow$ to represent a random assignment of a link $(i,j) \in \mathcal{E}_i$. Each agent attempts to expand its local

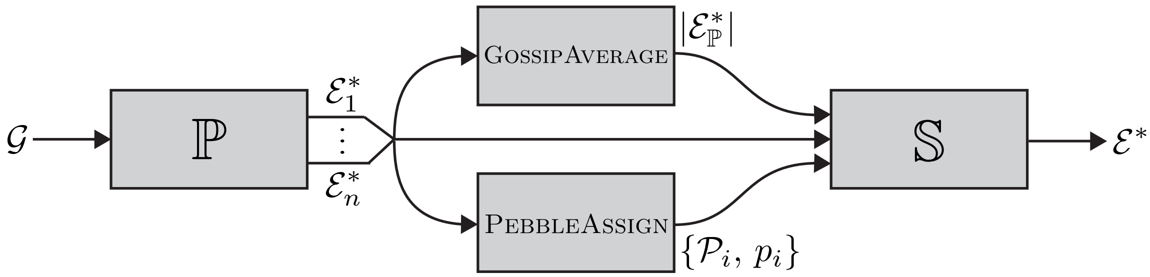


Fig. 6. Block diagram of the parallelized pebble game algorithm for decentralized planar rigidity evaluation. The parallel phase algorithm $\mathbb{P}$ acts as input for the serial phase $\mathbb{S}$, through localized pebble assignment and a gossip-based averaging.

independence set $\mathcal{E}_i^*$ by exploiting messaging to determine a neighbor j's feasibility as an endpoint in $\mathcal{G}^*(k)$, ensuring that the EER and TIER conditions are fulfilled. Thus, for each edge $e_i \triangleq (i,j)$ chosen randomly from $\mathcal{E}_i$, agent i requests neighbor j's commitment to $\mathcal{G}^*$ through message EDGEREQUESTMSG(i,j), shown in Algorithm 6, lines 3–4, and blocks further edge consideration, similar to the leader logic described in Section III-B. When all edges in $\mathcal{E}_i$ have been considered, line 8, the agent enters into the idle state until the network stopping condition is met. Algorithm termination then occurs when all agents have entered into the idle state. The remaining logic for independent edge selection and rule checking resides in the message handlers related to EDGEREQUESTMSG(i,j) of Algorithm 7, as will be discussed in the following.

C. Inter-Agent Messaging

The reception of an edge request message by agent i from neighbor $j \in \mathcal{N}_i$ triggers the message handling logic HANDLEEDGEREQUEST(i,j) depicted in Algorithm 7. Recall that when an agent i receives such a request from j it indicates an attempt by j to make the assignment of edge (j,i) to its independent set $\mathcal{E}_j^*$. Requests for an edge assignment must therefore be tested for fulfillment of the EER and TIER rules. However, as all agents are trying to add incident edges to the independent edge set simultaneously, it may be the case that two agents sharing an edge conflict on which agent takes the edge, specifically as both cannot. Thus, receiving agent i first determines whether a request represents a conflict over edge (i,j). If there is a conflict, agents i and j resolve the conflict as follows:

Remark 4.1 (Conflict resolution): We define an *edge conflict* in the execution of algorithm $\mathbb{P}$ as the state in which an agent $i \in \mathcal{I}$ has received an EDGEREQUESMSG(i,j) from $j \in \mathcal{N}_i$, where for i it holds that *requestedFrom*$(i) = j$. Intuitively, this state indicates that both i *and* j are attempting to add edge (i,j) to $\mathcal{E}^*$, a condition that would introduce inconsistencies in $|\mathcal{E}^*|$. In such scenarios we assume there exists a function RESOLVECONFLICT$(i,j) : \mathcal{E} \to \mathcal{I}$ indicating the conflict *winner*, such that

$$\text{RESOLVECONFLICT}(i,j) = \text{RESOLVECONFLICT}(j,i) \quad (8)$$

for all $i \neq j \in \mathcal{I}$. Now, resolving a conflict is simply a matter of first deciding which agent wins and receives the edge, and then ensuring that the losing agent does not take the edge. The winner of a conflict is decided according to the predetermined policy, RESOLVECONFLICT, which is chosen as a design parameter, and the loser is denied the edge by rejecting its edge request, lines 2–8. This is accomplished by saturating the commitments of the winning agent, such that the edge request of the losing agent cannot be fulfilled due to violation of the EER and TIER rules. In achieving condition (8), we could consider various options, e.g., balancing $|\mathcal{E}_i^*|$ and $|\mathcal{E}_j^*|$ in choosing the winner, applying a simple predetermined condition such as agent label, or perhaps a more complex method such as considering optimal assignments based on some cost or utility function over (i,j).

After properly handling potential conflicts over (i,j), receiver i then simply responds to j via EDGERESPONSEMSG$(i,j,$*response*$)$ indicating its current commitment to $\mathcal{E}^*$, increases its own commitment count to guarantee independence preservation, and removes (i,j) from $\mathcal{E}_i$ to avoid double checking, lines 8–13.

In coherence with the edge request logic, in receiving an EDGERESPONSEMSG$(i,j,$*response*$)$, an agent i acts according to Algorithm 8. Again, as an edge response conveys agent j's commitment to $\mathcal{E}^*$, it is validated against the EER and TIER rules, with success resulting in the assignment of (i,j) to $\mathcal{E}_i$ and an incrementing of *committed*(i) as i is now an endpoint of $\mathcal{G}^*$, lines 3–4. After independence preserving assignment, agent i simply moves to consider its next incident edge by choosing a random $(i,j) \in \mathcal{E}_i$, lines 7–8.

At the conclusion of algorithm $\mathbb{P}$, a distributed independent edge set $\mathcal{E}_{\mathbb{P}}^*$ has been computed across each $\mathcal{E}_i^*$. However, by the EER and TIER definitions, it must hold generally that $\mathcal{E}_{\mathbb{P}}^* \subseteq \mathcal{E}^*$. Thus, to fully determine $\mathcal{E}^*$ we generate a composite algorithm by passing the terminal state of $\mathbb{P}$ to the serialized algorithm $\mathbb{S}$ in order to apply global network information in completing $\mathcal{E}^*$. The initial conditions for the serialized algorithm are generated from the output of the parallel algorithm in a way that it is feasible that the serial algorithm itself had run up to that point. Thus, the output of the parallel algorithm must be shaped to mimic the conditions of the serialized algorithm. To do so we simply apply a network summing algorithm to determine the overall size of the independent edge set (i.e., the sum of the local set sizes), and then assign pebbles to cover the independent edges as required for the pebble game. Specifically, we first apply gossip averaging [41], to yield $|\mathcal{E}_{\mathbb{P}}^*|$. Then, local pebble assignments can easily be determined locally, as verified by Proposition A.8. This composite construction, which we will denote $\mathbb{P}+\mathbb{S}$, is illustrated in Fig. 6. An example execution of the parallel algorithm is depicted in Fig. 7.

D. Complexity Analysis

Towards the real-world applicability of our propositions, we have complexity that scales well with network size:

Proposition 4.1 ($\mathbb{P}+\mathbb{S}$ complexity): By construction, executions of the $\mathbb{P}+\mathbb{S}$ algorithm have *worst-case* $O(n^2)$ messaging complexity, and $O(n)$ storage scaling.

Proof: For the $\mathbb{P}$ portion of the $\mathbb{P}+\mathbb{S}$ algorithm, we have that each of n agents communicates with at most $n-1$ neighbors, yielding $O(n^2)$ messaging. The gossip averaging for determining $|\mathcal{E}^*_{\mathbb{P}}|$ exhibits $O(n \log n)$ messaging [41], while a local pebble assignment trivially requires $O(n^2)$ operations. Thus we have an overall worst-case message complexity of $O(n^2)$ for a $\mathbb{P}+\mathbb{S}$ as $\mathbb{S}$ also has $O(n^2)$ messaging by Proposition 3.2. The overall storage complexity follows directly from the fact that assignments to $\mathcal{E}^*$ can be made only locally by each agent $i \in \mathcal{I}$, and thus scales like $O(n)$. ■

Finally, to close we can also roughly evaluate the expected improvement provided by the partial parallelization due to $\mathbb{P}$:

Proposition 4.2 (Parallel identification): An execution of $\mathbb{P}$, applied to a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ (where we assume every $v_i \in \mathcal{V}$ is an endpoint in $\mathcal{E}$), with $n \geq 3$ must result in the terminal condition

$$|\mathcal{E}^*_{\mathbb{P}}(\bar{t}_{\mathbb{P}})| \triangleq \left| \bigcup_i \mathcal{E}^*_{\mathbb{P},i}(\bar{t}_{\mathbb{P}}) \right| \geq \left\lceil \frac{n-1}{2} \right\rceil + 1 \qquad (9)$$

where $\lceil\cdot\rceil$ is the standard *ceiling operator*, yielding a lower bound on the independent edges identified by the $\mathbb{P}$ algorithm.

Proof: First, notice that we disregard the case of $n=1$ (as $|\mathcal{E}|=0$), and for $n=2$ we can always identify the single member of $\mathcal{E}^*$ due to symmetric conflict resolution. Now, observing that the single $n=2$ graph is a worst-case in terms of detectable independent edges , with $|\mathcal{E}^*|=\lceil n/2\rceil$, for $n \geq 3$ we can construct a similar worst-case graph by appending a single node and edge to the $n-1$ worst-case, as the appended edge will always be detectable by an EER operation. As we add a single detectable edge to the previous worst-case, a simple inductive argument yields the result. ■

The above result states directly that we can always detect a spanning tree over $\mathcal{G}$ using our proposed parallel and distributed interactions. Further, the number of edges that the parallelization identifies directly affects the number of edges left to be found, and thus the complexity of the serial execution. A summary of the advantages of our parallelization are given in the following remark, while technical analysis and detailed pseudocode can be found in the Appendix.

Remark 4.2 (Parallelization Benefits): First, as the parallel algorithm identifies at least a spanning tree in its execution by Proposition 4.2, it can quickly help to determine when the network is non-rigid without having to run the serialized step, yielding significant speed advantages in those scenarios. Additionally, as the parallel algorithm takes advantage of independence preserving operations, one could leverage it to build a rigid network autonomously and efficiently, e.g., in constructing a localizable sensor network topology. From a practical perspective, the parallelization is simply a more intelligent use of available network resources, i.e., it can be implemented without any additional communication or hardware requirements, so even a factor of two speedup may be convenient in practice. Also, such speedups relate well to realistic scenarios such as the time scales of external network influences and the speed of rigidity evaluation (see Section V-C). Even constant factor speedups can expand the applicability of our algorithms under faster switching topologies or environmental conditions.

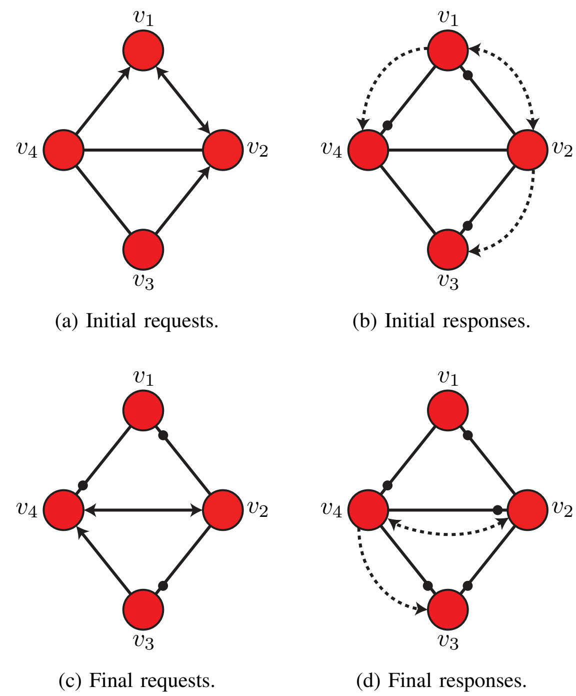


Fig. 7. Parallel messaging for a minimally rigid graph with $n=4$. Inter-agent requests are denoted by solid arrows, responses by dashed arrows, and pebble assignments by solid dots. Notice that conflicts occur in (a) over (v_1,v_2) and in (c) over (v_2,v_4), with resolution dictated by agent label $i<j$ for $j \in \mathcal{N}_i$.

V. Simulation Results

A. A Rigidity Control Scenario

We wish to demonstrate here a scenario where network rigidity can be controlled in a dynamic multi-robot system. To begin, consider that in controlling generic rigidity we need only to disallow the loss of independent edges $(i,j) \in \mathcal{E}^*$. For this purpose, we can employ the *constrained interaction framework* proposed by Williams and Sukhatme in [4], a very brief overview of which will be given here. Assuming proximity-limited sensing and communication, together with agent dynamics $\dot{x}_i = u_i$, the constrained interaction framework regulates link addition and deletion *spatially*, through a switching combination of hysteresis, attraction, and repulsion, in order to satisfy a desired set of constraints. In particular, each agent is assigned *predicates* $P^a_{ij}, P^d_{ij} : \mathcal{V}\times\mathcal{V} \leftrightarrow \{0,1\}$ that indicate constraint violations if link (i,j) were gained or lost. Thus, in applying our proposed decentralized pebble game

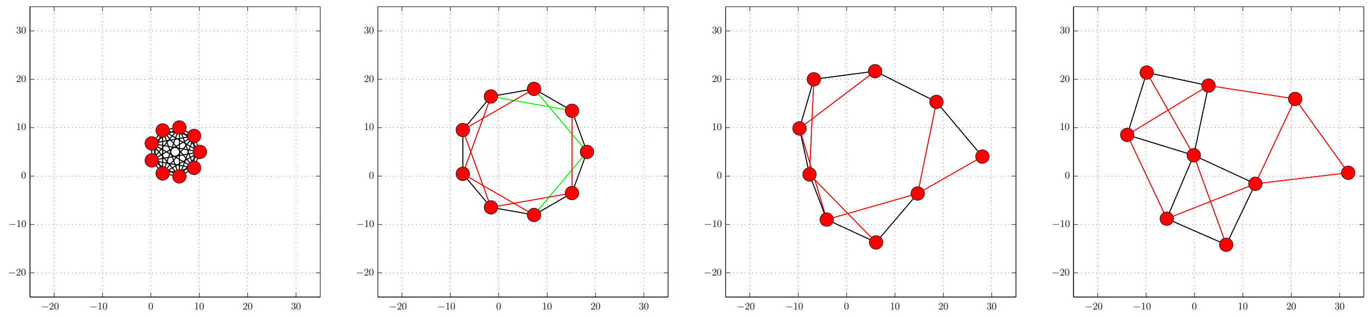


Fig. 8. Rigidity control simulation for $n = 9$ mobile agents applying a dispersive objective with a rigidity maintenance constraint: (a) initial rigid configuration; (b) and (c) intermediate configurations; (d) final converged and rigid configuration. Green edges are not independent and thus can be lost, while red edges are actively retained due to their independence.

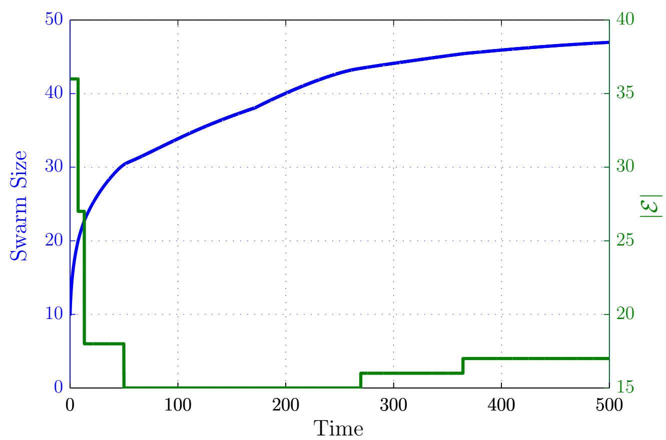


Fig. 9. Swarm size and edge set cardinality for the rigidity control simulation.

to identify local sets $\mathcal{E}_i^* \, \forall \, i \in \mathcal{I}$, we arrive directly at rigidity preserving predicates. That is,

$$P_{ij}^a \triangleq 0, \quad P_{ij}^d \triangleq (i,j) \in (\mathcal{E}_i^* \cup \mathcal{E}_j^*) \tag{10}$$

Now, we are prepared to present our rigidity control simulation results. We assume a system of $n = 9$ mobile agents, each with proximity-limited communication and sensing, applying for the sake of link deletion, a dispersive objective controller, yielding agent controllers with generic form:

$$u_i = u_{CI} - \nabla_{x_i} \left(\sum_{i \in \mathcal{N}_i} \frac{1}{\|x_i - x_j\|^2} \right) \quad \forall \, i \in \mathcal{I} \tag{11}$$

where u_{CI} is the control contribution due to the constrained interaction framework and predicates (10). The agents begin in the fully connected initial configuration given by the *ring network* depicted in Fig. 8a, satisfying the initial condition $\mathcal{G}(0) \in \mathcal{G}_{\mathbb{R}}$. Through controllers (11) the agents reach intermediate configurations given by Figs. 8b and 8c, ultimately terminating in the final configuration in Fig. 8d. Fig. 9 depicts the spatial size of the swarm, i.e., the largest distance between any two agents, and the size of the independent edge set, which dictates network rigidity. Thus Fig. 9 demonstrates that the dispersive objective is achieved through increasing swarm size, and that the network remains rigid as the size of the independent set is bounded below by $2n - 3$.

B. Contiki Implementation: Real-World Feasibility Results

To determine the performance of our algorithms under realistic networking conditions, we consider the *Contiki* operating system, together with the *Cooja* network simulator[3]. We implemented both the serial and parallel decentralized pebble game for Contiki and tested our codebase against a range of emulated hardware platforms and communication stacks for correctness. A Monte Carlo set was simulated by generating rigid and non-rigid networks for $n \in \{5, 29\}$, yielding the results depicted in Fig. 10. Specifically, Fig. 10 (top) compares the execution time (seconds) for the serial and parallel algorithms, while Fig. 10 (bottom) shows the per-agent messaging burden. It is clear that both of our algorithms exhibit feasible and efficient performance, with actual scaling that is approximately $O(n)$ in both execution and messaging. The parallel version however represents our goal of real-world capability by outperforming the serial version by a factor 2, as even in reasonably sized networks, execution times for evaluation are under 1 second. An initial version of the base code for our proposed algorithms has also been released for application in the robotics community[4].

C. Realistic Considerations

Realistic applications present difficult and unpredictable influences, e.g., wind gusts or variability in ocean currents in autonomous surface vehicles (ASVs). These environmental variables and their timescales will directly impact the capability of the network to determine network rigidity in a timely fashion, as the network topology may change too often due to uncontrollable influences. Thus, the relationship between the parameters of an application, the properties of the employed communication network, and the numerical bounds on rigidity evaluation run-time must all be well understood by an implementer. Informally, the switching time of the topology, which is dictated by communication hardware and application-specific factors, and the network size become crucial design variables as they determine the feasibility of computing network rigidity in time to steer the network away from non-rigidity. In general, networks which exhibit short switching times will

[3]See http://www.contiki-os.org.

[4]See http://github.com/Attilio-Priolo/Rigidity_Check_Contiki

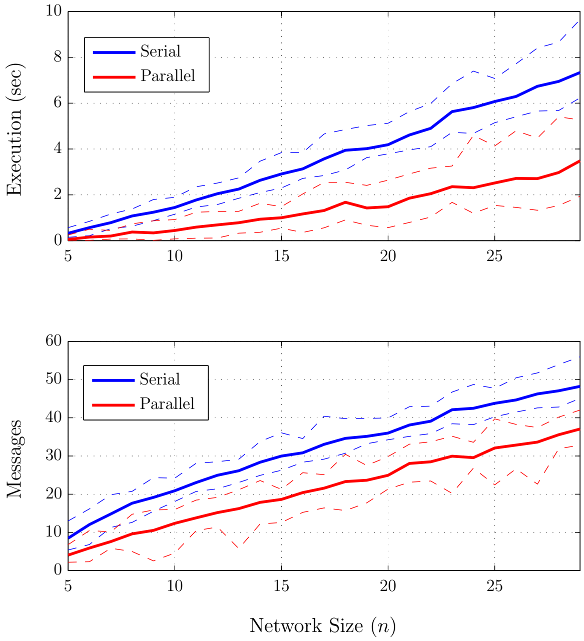


Fig. 10. Monte Carlo results from a Contiki networking environment demonstrating execution time and message complexity for the $\mathbb{S}$ and $\mathbb{P}+\mathbb{S}$ algorithms. Solid lines are average values, while dashed lines are maximum and minimums.

require smaller network size or faster communication to combat external influences for feasible operation.

VI. Conclusions and Future Work

In this paper, we considered the problem of evaluating the rigidity of a planar network under the requirements of decentralization, asynchronicity, and parallelization. We proposed the decentralization of the pebble game algorithm of Jacobs et. al., based on asynchronous inter-agent message-passing and distributed auctions for electing network leaders. Further, we provided a parallelization of our methods that yielded significantly reduced execution time and messaging burden. Finally, we provided a simulated application in decentralized rigidity control, and Monte Carlo analysis of our algorithms in a *Contiki* networking environment, illustrating the real-world applicability of our methods.

Directions for future work include demonstrating our methods on robotic hardware in field experiments, extensions to global rigidity for even stronger guarantees in multi-robot applications, and the inclusion of network utility metrics to yield decentralized rigidity evaluation and control with provable optimality conditions.

Appendix

A. The Serialized Algorithm

To complement our discussion of the $\mathbb{S}$ algorithm, we analyze the correctness, finite termination, and cost properties of our algorithms. First, we formally establish the stopping condition for the $\mathbb{S}$ algorithm:

Definition A.1 ($\mathbb{S}$ stopping condition): As previously discussed, the $\mathbb{S}$ algorithm terminates upon satisfaction of the following condition:

$$f_{stop}^{\mathbb{S}} \triangleq \left\{ \left(\sum_{i=1}^{n} b_i = 0 \right) \vee \left(\sum_{i=1}^{n} |\mathcal{E}_i^*| = 2n - 3 \right) \right\} \quad (12)$$

where the $\sum_i b_i = 0$ indicates that all agents have been a leader, and $\sum_i |\mathcal{E}_i^*| = 2n-3$ is detected by the lead agent on line 24 of Algorithm 2.

Next, we verify that our formulation guarantees the *entire* network is evaluated for rigidity, with *no* edge reconsideration, and further that mutual exclusion of the local independent sets holds:

Proposition A.1 (Edge consideration, mutual exclusion): Disregarding algorithm termination when $|\mathcal{E}^*| = 2n-3$, every $(i,j) \in \mathcal{E}$ is eligible to be considered for independence. Further, $\mathcal{E}_i^* \cap \mathcal{E}_j^* = \emptyset$ holds for all $i \neq j \in \mathcal{I}$.

Proof: These results are a simple consequence of the guaranteed convergence of auction (4), $b_i = 0$ for all *beenLeader*$(i) = 1$ guaranteeing no reelection, and the initialization of $\mathcal{E}_i$ with edges *not* shared with previous leaders. ∎

To ensure timely results, we must also have finite termination of $\mathbb{S}$:

Proposition A.2 ($\mathbb{S}$ termination): Consider the execution of the $\mathbb{S}$ algorithm as described in Sections III-A, III-B, and III-C. By construction, it follows that the stopping condition $f_{stop}^{\mathbb{S}}$ of (12) is satisfied after a *finite* number of clock ticks.

Proof: We can guarantee no message-induced race conditions by Assumptions 2 and 3, and that there exist no algorithmic race conditions due to the internal blocking on line 2 of Algorithm 2. From the request checking mechanism (line 2 Algorithm 3) and the guaranteed delivery of inter-agent messages by the best-effort Assumption 3, we have that all pebble request messaging rooted at agent i with *isLeader*$(i) = 1$, is finite, i.e., every pebble request receives a response. Now, the finiteness of execution is a direct consequence of the finiteness of each $\mathcal{E}_i, \forall i \in \mathcal{I}$ and the finite convergence of auction (4) [40], as there exists no leader reelection by construction. ∎

Now we come to our primary result concerning the correctness of the $\mathbb{S}$ algorithm:

Proposition A.3 ($\mathbb{S}$ correctness): Consider an execution of $\mathbb{S}$, applied to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. It follows that by construction we are guaranteed to identify $|\mathcal{E}^*| = 2n-3$ independent edges when $\mathcal{G} \in \mathcal{G}_{\mathbb{R}}$, and $|\mathcal{E}^*| < 2n-3$ otherwise, i.e., $\mathbb{S}$ properly identifies the generic rigidity of $\mathcal{G}$.

Proof: First, notice that by Proposition A.1, we can ensure that every $(i,j) \in \mathcal{E}$ is eligible for quadrupling and pebble covering as dictated by the original pebble game [28], and further that $\mathcal{E}_i^* \cap \mathcal{E}_j^* = \emptyset$ holds for all $i \neq j \in \mathcal{I}$ ensures that $|\mathcal{E}^*|$ is properly tracked by our distributed storage. Thus, correctness is shown by arguing that our leadership and messaging formulation is faithful to the rules of the pebble game. This result follows by observing that pebble assignments and shift operations are only made locally (line 8 of Algorithm 2, line 7 of Algorithm 3, and lines 2, 4, and 8 of Algorithm 4), and that the pebble search mechanism respects the network's distributed pebble assignments $\mathcal{P}_i \, \forall i \in \mathcal{I}$ (line 12 of Algorithm

Algorithm 2 Leader execution logic.

1: **procedure** LEADERRUN(i)
2: **while** $e_i \triangleq (i,j)$ **do** ▷ Continue pebble covering
3: **while** *Quadrupled Copies* ≤ 4 **do**
4: **if** $p_i > 0$ **then** ▷ Assign local pebble
5: $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup e_i$
6: $p_i \leftarrow p_i - 1$
7: **else** ▷ Request pebble along first edge
8: PEBBLEREQUESTMSG(i, $\mathcal{P}_i(1,2)$)
9: **return**
10: **end if**
11: **end while**
12: ▷ Quadrupling success, return 3 pebbles:
13: $\mathcal{P}_i \leftarrow \emptyset$
14: $p_i \leftarrow 2$
15: Return 1 pebble to v_j
16: ▷ Add independent edge and check rigidity:
17: $\mathcal{E}_i^* \leftarrow \mathcal{E}_i^* \cup e_i$
18: **if** $|\mathcal{E}^*| = 2n-3$ **then**
19: Send network rigidity notification
20: **return**
21: **end if**
22: ▷ Go to next incident edge:
23: $\mathcal{E}_i \leftarrow \mathcal{E}_i - e_i$
24: $e_i \leftarrow (i,j) \in \mathcal{E}_i$
25: **end while**
26: ▷ All local edges checked:
27: Initiate leadership transfer auction
28: **end procedure**

Algorithm 3 Pebble request handler for agent i.

1: **procedure** HANDLEPEBBLEREQUEST(*from*, i)
2: **if** *Request Not Unique* **then** ▷ Already requested
3: PEBBLENOTFOUNDMSG(i, *from*)
4: **return**
5: **end if**
6: **if** $p_i > 0$ **then** ▷ Local pebble available
7: $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup (i, \mathit{from})$ ▷ Shift free pebble
8: $p_i \leftarrow p_i - 1$
9: PEBBLEFOUNDMSG(i, *from*)
10: **else** ▷ Request along first assigned edge
11: PEBBLEREQUESTMSG(i, $\mathcal{P}_i(1,2)$)
12: *requester*(i) $\leftarrow$ *from*
13: **end if**
14: **end procedure**

2, line 11 of Algorithm 3, and line 3 of Algorithm 5). Finally, as there is only one leader active at any time, each quadrupling operation (lines 6-17, Algorithm 2) is sound with respect to the current set $\mathcal{E}^*$, and the result follows. ■

The above result demonstrates that $\mathbb{S}$ is sound in terms of planar rigidity evaluation.

B. The Parallelized Algorithm

We begin with a proof of the independence preservation of the EER and TIER graph operations:

Algorithm 4 Pebble found handler for agent i.

1: **procedure** HANDLEPEBBLEFOUND(*from*, i)
2: $\mathcal{P}_i \leftarrow \mathcal{P}_i - (i, \mathit{from})$ ▷ Free local pebble
3: **if** *isLeader*(i) **then**
4: $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup e_i$ ▷ Expand covering
5: **else** ▷ Give free pebble to requester
6: $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup (i, \mathit{requester}(i))$
7: PEBBLEFOUNDMSG(i, *requester*(i))
8: **end if**
9: **end procedure**

Algorithm 5 Pebble not found handler for agent i.

1: **procedure** HANDLEPEBBLENOTFOUND(*from*, i)
2: **if** *Paths Searched* < 2 **then** ▷ Search other path
3: PEBBLEREQUESTMSG(i, $\mathcal{P}_i(2,2)$)
4: **else** ▷ Search failed, no free pebbles
5: **if** *isLeader*(i) **then** ▷ e_i is redundant
6: Return pebbles assigned to e_i
7: ▷ Go to next incident edge:
8: $\mathcal{E}_i \leftarrow \mathcal{E}_i - e_i$
9: $e_i \leftarrow (i,j) \in \mathcal{E}_i$
10: **else**
11: PEBBLENOTFOUNDMSG(i, *requester*(i))
12: **end if**
13: **end if**
14: **end procedure**

Proposition A.4 (Independence preservation): Consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ having edges $\mathcal{E}$ forming an independent set according to Definition 2.2. The edge addition operations $[\cdot]_e^+$ over $\mathcal{G}$ abiding by the EER and TIER requirements of Definitions 4.3 and 4.4 are independence preserving in the sense of Definition 4.2, respectively. Further, considering a sequence of graphs $\{\mathcal{G}(0), \ldots, \mathcal{G}(m)\}$ generated by

$$\mathcal{G}(0) = \mathcal{G} \quad \mathcal{G}(k) = [\mathcal{G}(k-1)]_e^+, \; k = 1, \ldots, m \tag{13}$$

over EER and TIER operations yields graph $\mathcal{G}^m$ having independent edges $\mathcal{E}^m$.

Proof: First, consider the case of an EER operation over $\mathcal{G}$. By the independence of $\mathcal{E}$, we have by Definition 2.2 that for every subgraph $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$, $|\bar{\mathcal{E}}| \leq 2|\bar{\mathcal{V}}| - 3$. In the augmented graph edge e introduces expanded subgraphs containing e all having the property

$$|\bar{\mathcal{E}}| + 1 \leq 2|\bar{\mathcal{V}}| - 3 + 1 \leq 2(|\bar{\mathcal{V}}| + 1) - 3 \tag{14}$$

due to the node expansion property of the EER, all of which therefore abide by the independence subgraph property. The remaining subgraphs of $\mathcal{G}^+$ are independent by assumption. Thus, we conclude that the EER operation according to Definition 4.3 is independence preserving.

Now, consider the application of the TIER operation over $\mathcal{G}$. By Definition 4.4 there must exist an endpoint of $e \triangleq (i,j)$, indexed by $i \in \mathcal{I}$, with exactly $\mathcal{N}_i = 1$ over $\mathcal{G}$. Thus, we can view the edges (i,j) and (i,k) with $k \in \mathcal{N}_i$, and the node i as members of a two edge Henneberg operation, as in Section 3 of [27], e.g., adding vertex v_4 with edges (4,2) and (4,1) in

Algorithm 6 Parallel execution logic for agent i.

1: **procedure** PARALLELRUN(i)
2:   **if** $e_i \triangleq (i,j) \neq \mathbf{0}$ **then** $\triangleright$ Next incident edge
3:     EDGEREQUESTMSG(i, j)
4:     *requestedFrom*$(i) \leftarrow j$
5:     **return**
6:   **end if**
7:   $\triangleright$ All local edges checked:
8:   *idle*$(i) \leftarrow$ Yes
9: **end procedure**

Algorithm 7 Parallel edge request handler for agent i.

1: **procedure** HANDLEEDGEREQUEST(i, j)
2:   *response* $\leftarrow$ *committed*(i)
3:   **if** *requestedFrom*$(i) = j$ **then** $\triangleright$ Edge contention
4:     **if** RESOLVECONFLICT(i, j) $= i$ **then**
5:       *response* $\leftarrow 2$ $\triangleright$ Ensure i wins edge
6:     **end if**
7:   **end if**
8:   EDGERESPONSEMSG(i, j, *response*)
9:   **if** *response* < 2 **then** $\triangleright$ Max of 2 incident edges
10:     *committed*$(i) \leftarrow$ *committed*$(i) + 1$
11:   **end if**
12:   $\triangleright$ Do not double check (i,j):
13:   $\mathcal{E}_i \leftarrow \mathcal{E}_i - (i,j)$
14: **end procedure**

Algorithm 8 Parallel edge response handler for agent i.

1: **procedure** HANDLEEDGERESPONSE(i, j, *response*)
2:   **if** *response* < 2 **then** $\triangleright$ Independence guaranteed
3:     $\mathcal{E}_i^* \leftarrow \mathcal{E}_i^* \cup (i,j)$
4:     *committed*$(i) \leftarrow$ *committed*$(i) + 1$
5:   **end if**
6:   $\triangleright$ Go to next incident edge:
7:   $\mathcal{E}_i \leftarrow \mathcal{E}_i - (i,j)$
8:   $e_i \not\leftarrow (i,j) \in \mathcal{E}_i$
9: **end procedure**

Fig. 5b. As the edge subtraction operation $[\cdot]_e^-$ is independence preserving, the graph $\mathcal{G}^+$ described by applying the previous two edge Henneberg operation to $[\mathcal{G}]_{(i,k)}^-$ has independent edges by Proposition 3.1 of [27]. Briefly, this result follows from the independence of $\mathcal{E}$ and the relationships, $|\mathcal{E}| = |\mathcal{E}^+| - 2$ and $|\mathcal{V}| = |\mathcal{V}^+| - 1$. Thus, the TIER operation is also independence preserving.

Finally, the independence preservation of a sequence of EER and TIER operations is a trivial consequence of the initial independence of $\mathcal{E}$ and the IP properties of each edge augmentation. ∎

To complement our discussion of the $\mathbb{P} + \mathbb{S}$ algorithm, we analyze the correctness, finite termination, and cost properties of our algorithms. First, we verify that $\mathcal{E}_{\mathbb{P}}^* \triangleq \bigcup_i \mathcal{E}_{\mathbb{P},i}^*$ has a valid distributed construction:

Proposition A.5 (Parallel mutual exclusion): Consider the application of the parallel $\mathbb{P}$ algorithm to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. It follows that upon termination we have mutual exclusion $\mathcal{E}_i^* \cap \mathcal{E}_j^* = \emptyset, \forall\, i \neq j \in \mathcal{I}$.

Proof: Given the assumptions of asynchronicity in messaging and the FIFO queuing of received messages (Assumption 3) and execution devoid of race conditions (Assumption 2), the following scenarios must be considered, viewed from the instant when agent i handles an EDGERESPONSEMSG from j, implying that $e_i = (i,j)$ and $(i,j) \notin \mathcal{E}_i^*$ (line 2, Algorithm 8):

- $(i,j) \notin \mathcal{E}_j^*$: We must consider two cases here, either $e_j = (i,j)$ or $e_j \neq (i,j)$. First, in the trivial case of $e_j \neq (i,j)$, it follows from reception of an EDGERESPONSEMSG from j, the atomic nature of execution, and line 13 of Algorithm 7, that regardless of assignment to $\mathcal{E}_i^*$, $(i,j) \notin \mathcal{E}_j^*$ for all execution $t > 0$. When $e_j = (i,j)$, the conflict resolution of line 3-7 in Algorithm 7, together with line 2 of Algorithm 8 ensures that simultaneous requests made over (i,j) agree on assignment, specifically as by assumption RESOLVECONFLICT$(i,j) =$ RESOLVECONFLICT(j,i).
- $(i,j) \in \mathcal{E}_j^*$: Here it is implied that at some previous time agent i received and responded to a EDGEREQUESTMSG from j. As agent i being in a state of response reception over (i,j) is contradictory given line 13 of Algorithm 7, it must be the case that requests over (i,j) have been made in concert. However, as previously stated, the conflict resolution ensures $\mathcal{E}_i^* \cap \mathcal{E}_j^* = \emptyset$ in such scenarios.

Notice that due to the uniformity of execution and messaging logic across $i \in \mathcal{I}$, the previous scenarios hold equivalently from the perspective of agent j, and thus for all pairs $\{i \neq j \,|\, (i,j) \in \mathcal{E}\}$, and the result follows. ∎

Of course, $\mathcal{E}_{\mathbb{P}}^*$ must also fulfill the independence requirements of Definition 2.2, as is shown below.

Proposition A.6 (Parallel Correctness): Consider the algorithm $\mathbb{P}$ applied to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For all execution $t > 0$ it follows that edge addition operations, $\mathcal{E}_i^*(t^+) = \mathcal{E}_i^*(t) \cup (i,j)$ (line 3 Algorithm 8), are independence preserving and $\bigcup_i \mathcal{E}_i^*$ is independent with $|\bigcup_i \mathcal{E}_i^*| \leq 2n - 3$.

Proof: Resting again on Assumptions 2 and 3, and the uniform conflict resolution of RESOLVECONFLICT(i,j), this result is a consequence of the commitment counting rules (lines 10 and 5 of Algorithms 7 and 8), and the condition on line 2 of Algorithm 8 that enforces the cardinality of endpoint j in $\bigcup_i \mathcal{E}_i^*(t^+)$. In particular, when $j \notin \bigcup_i \mathcal{E}_i^*(t)$ (*committed*$(j) = 0$), $\mathcal{E}_i^*(t^+) = \mathcal{E}_i^*(t) \cup (i,j)$ constitutes an EER operation (c.f. Definition 4.3), otherwise when $j \in \bigcup_i \mathcal{E}_i^*(t)$ (*committed*$(j) = 1$), $\mathcal{E}_i^*(t^+) = \mathcal{E}_i^*(t) \cup (i,j)$ constitutes a TIER operation (c.f. Definition 4.4). As sequences of EER and TIER operations preserve independence by Proposition A.4, $\bigcup_i \mathcal{E}_i^*$ is independent, with $|\bigcup_i \mathcal{E}_i^*| \leq 2n - 3$ following directly from the Laman conditions Theorem 2.1. ∎

To ensure timely results, we must also have finite termination of $\mathbb{P} + \mathbb{S}$:

Proposition A.7 ($\mathbb{P}$ termination): Consider the execution of the $\mathbb{P}$ algorithm as described in Section IV-B. By construction, it follows that the stopping condition, i.e., all agents are idle, is satisfied after a *finite* number of clock ticks.

Proof: We can again guarantee no message-induced race conditions by Assumptions 2 and 3. Thus, the finiteness of execution is a direct consequence of the finiteness of each $\mathcal{E}_i$, $\forall\, i \in \mathcal{I}$, the internal blocking on line 2 of Algorithm 6, the guaranteed delivery of inter-agent messages by the best-effort Assumption 3, and finally the symmetric conflict resolution of Remark 4.1, disallowing conflict based race conditions. ■

To constitute a valid initial condition for $\mathbb{S}$, the pebble assignments applied to the terminal state $\mathcal{E}_{\mathbb{P}}^*$ of $\mathbb{P}$ must be sound:

Proposition A.8 (Pebble Assignments): Consider an execution of $\mathbb{P}$, applied to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. There must exist a *local* pebble covering for every $(i, j) \in \mathcal{E}_i^* \,\forall\, i \in \mathcal{I}$, that is a *local* assignment of a pebble by either i or j to (i, j).

Proof: In guaranteeing that such an assignment exists, we rely on the properties of the EER and TIER operations. As each operation $\mathcal{E}_i^*(t^+) = \mathcal{E}_i^*(t) \cup (i, j)$ respects Proposition A.4 by Proposition A.6, we have that for any $(i, j) \in \bigcup_i \mathcal{E}_i^*$ there must exist an endpoint i or j with at most two incident edges. From this endpoint we can thus always select a pebble to cover (i, j) as each agent $i \in \mathcal{I}$ is initially assigned two pebbles. ■

Now we come to our primary result concerning the correctness and finite termination of the $\mathbb{P} + \mathbb{S}$ algorithm:

Proposition A.9 ($\mathbb{P} + \mathbb{S}$ correctness and termination): Consider an execution of $\mathbb{P} + \mathbb{S}$, applied to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. It follows that the terminal system state:

$$\mathbb{P}(\bar{t}_{\mathbb{P}}) \triangleq \left\{ \bigcup_i \mathcal{E}_i^*(\bar{t}_{\mathbb{P}}), \mathcal{P}_i, p_i \right\} \quad \forall\, i \in \mathcal{I} \tag{15}$$

is a valid initial condition for the $\mathbb{S}$ algorithm, the execution of $\mathbb{P} + \mathbb{S}$ terminates after a *finite* number of clock ticks, and properly identifies the generic topology rigidity of $\mathcal{G}$.

Proof: First, we have directly from Propositions A.2 and A.7, the known finite convergence of gossip averaging [41], and the trivial finiteness of a local pebble assignment process, that the composite execution of $\mathbb{P}+\mathbb{S}$ terminates in finite time. Now, from Propositions A.5 and A.6 it follows that the state $\mathbb{P}(\bar{t}_{\mathbb{P}})$ represents a properly distributed and independent edge set, and from Proposition A.8 that there must exist pebble assignments $\mathcal{P}_i$ that are local shift operations relative to Definition 2.3, and thus constitute a valid pebble covering. Thus the application of $\mathbb{S}$ with input $\mathbb{P}(\bar{t}_{\mathbb{P}})$ is correct by Proposition A.3, and the result follows. ■

REFERENCES

[1] N. E. Leonard, D. A. Paley, F. Lekien, R. Sepulchre, D. M. Fratantoni, and R. E. Davis, "Collective Motion, Sensor Networks, and Ocean Sampling," *Proceedings of the IEEE*, 2007.
[2] M. A. Hsieh, E. Forgoston, T. W. Mather, and I. B. Schwartz, "Robotic manifold tracking of coherent structures in flows," in *IEEE International Conference on Robotics and Automation*, 2012.
[3] J. Cortes, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, 2004.
[4] R. K. Williams and G. S. Sukhatme, "Constrained Interaction and Coordination in Proximity-Limited Multi-Agent Systems," *IEEE Transactions on Robotics*, 2013.
[5] ——, "Topology-Constrained Flocking in Locally Interacting Mobile Networks," in *IEEE International Conference on Robotics and Automation*, 2013.
[6] S. Martínez, J. Cortes, and F. Bullo, "Motion Coordination with Distributed Information," *Control Systems Magazine, IEEE*, 2007.
[7] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, 2004.
[8] S. Kar and J. Moura, "Distributed Consensus Algorithms in Sensor Networks With Imperfect Communication: Link Failures and Channel Noise," *IEEE Transactions on Signal Processing*, 2009.
[9] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, 2005.
[10] R. Olfati-Saber and R. M. Murray, "Graph rigidity and distributed formation stabilization of multi-vehicle systems," in *IEEE Conference on Decision and Control*, 2002.
[11] T. Eren, P. N. Belhumeur, and A. Morse, "Closing ranks in vehicle formations based on rigidity," in *IEEE Conference on Decision and Control*, 2002.
[12] B. Anderson, C. Yu, B. Fidan, and J. Hendrickx, "Rigid graph control architectures for autonomous formations," *Control Systems, IEEE*, 2008.
[13] A. N. Bishop, I. Shames, and B. D. Anderson, "Stabilization of rigid formations with direction-only constraints," in *IEEE Conference on Decision and Control and European Control Conference*, 2011.
[14] S. Motevallian, C. Yu, and B. D. O. Anderson, "Multi-agent rigid formations: A study of robustness to the loss of multiple agents," in *IEEE Conference on Decision and Control and European Control Conference*, 2011.
[15] T. Eren, "Formation shape control based on bearing rigidity," *International Journal of Control*, 2012.
[16] T. Eren, W. Whiteley, A. Morse, P. N. Belhumeur, and B. D. O. Anderson, "Sensor and network topologies of formations with direction, bearing, and angle information between agents," in *IEEE Conference on Decision and Control*, 2003.
[17] T. Eren, W. Whiteley, and P. N. Belhumeur, "Using Angle of Arrival (Bearing) Information in Network Localization," in *Decision and Control, 2006 45th IEEE Conference on*, 2006.
[18] J. Aspnes, T. Eren, D. K. Goldenberg, A. Morse, W. Whiteley, Y. R. Yang, B. D. O. Anderson, and P. N. Belhumeur, "A Theory of Network Localization," *IEEE Transactions on Mobile Computing*, 2006.
[19] I. Shames, A. N. Bishop, and B. D. O. Anderson, "Analysis of Noisy Bearing-Only Network Localization," *IEEE Transactions on Automatic Control*, 2013.
[20] A. R. Berg and T. Jordan, "A proof of Connelly's conjecture on 3-connected circuits of the rigidity matroid," *J. Comb. Theory Ser. B*, 2003.
[21] B. Jackson and T. Jordan, "Connected rigidity matroids and unique realizations of graphs," *J. Comb. Theory Ser. B*, 2005.
[22] B. Hendrickson, "Conditions for unique graph realizations," *SIAM J. Comput.*, 1992.
[23] G. Laman, "On graphs and rigidity of plane skeletal structures," *Journal of Engineering Mathematics*, 1970.
[24] L. Asimow and B. Roth, "The rigidity of graphs, II," *Journal of Mathematical Analysis and Applications*, 1979.
[25] B. Roth, "Rigid and Flexible Frameworks," *The American Mathematical Monthly*, 1981.
[26] L. Lovasz and Y. Yemini, "On Generic Rigidity in the Plane," *Siam Journal on Algebraic and Discrete Methods*, 1982.
[27] T.-S. Tay and W. Whiteley, "Generating Isostatic Frameworks," *Structural Topology*, 1985.
[28] D. J. Jacobs and B. Hendrickson, "An algorithm for two-dimensional rigidity percolation: the pebble game," *J. Comput. Phys.*, 1997.
[29] D. Zelazo and F. Allgower, "Growing optimally rigid formations," in *American Control Conference*, 2012.
[30] R. Ren, Y.-Y. Zhang, X.-Y. Luo, and S.-B. Li, "Automatic generation of optimally rigid formations using decentralized methods," *Int. J. Autom. Comput.*, 2010.
[31] D. Zelazo, A. Franchi, F. Allgower, H. H. Bulthoff, and P. R. Giordano, "Rigidity Maintenance Control for Multi-Robot Systems," in *Robotics: Science and Systems*, 2012.
[32] B. Fidan, J. M. Hendricks, and B. D. O. Anderson, "Edge contraction based maintenance of rigidity in multi-agent formations during agent loss," in *Control and Automation, 2009. MED '09. 17th Mediterranean Conference on*, 2009.
[33] R. K. Williams, A. Gasparri, A. Priolo, and G. S. Sukhatme, "Distributed Combinatorial Rigidity Control in Multi-Agent Networks," in *IEEE Conference on Decision and Control*, 2013.
[34] ——, "Decentralized Generic Rigidity Evaluation in Interconnected Systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[35] A. Nedic, "Asynchronous Broadcast-Based Convex Optimization over a Network," *IEEE Transactions on Automatic Control*, 2010.
[36] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005.
[37] R. Olfati-Saber, J. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proceedings of the IEEE*, 2007.
[38] H. Gluck, "Almost all simply connected closed surfaces are rigid," in *Geometric Topology*, 1975.
[39] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, 1988.
[40] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas, "A distributed auction algorithm for the assignment problem," in *IEEE Conference on Decision and Control*, 2008.
[41] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip Algorithms for Distributed Signal Processing," *Proceedings of the IEEE*, 2010.
[42] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *Information Theory, IEEE Transactions on*, 2006.

Ryan K. Williams received the B.S. degree in computer engineering from Virginia Polytechnic Institute and State University in 2005. He is currently a Ph.D. candidate in electrical engineering at the University of Southern California, and a member of the Robotic Embedded Systems Laboratory. His current research interests include control, cooperation, and intelligence in distributed multi-agent systems, topological methods in cooperative phenomena, and distributed algorithms for optimization, estimation, inference, and learning. Ryan K. Williams is a Viterbi Fellowship recipient, has been featured by various news outlets, including the L.A. Times, and has a patent pending for his work on high-speed autonomous underwater vehicles.

Andrea Gasparri received the Graduate degree cum laude in Computer Science in 2004 and the Ph.D. degree in Computer Science and Automation in 2008, both from the University of Roma Tre, in Rome Italy. He is currently an Assistant Professor at the Department of Engineering, University of Roma Tre, in Rome, Italy. His current research interests include mobile robotics, sensor networks, and, more generally, networked multi-agent systems. He has been awarded with the Italian grant FIRB Futuro in Ricerca 2008, and project NECTAR (NEtworked Collaborative Team of Autonomous Robots) funded by the Italian Ministry of Research and Education (MIUR). He has been a visiting researcher at institutions including Universite Libre de Bruxelles (ULB) - Belgium, City College of New York (CCNY) - USA, and University of Southern California (USC) - USA.

Attilio Priolo received the Master degree in Computer Science and Automation Engineering in 2009 and the Ph.D degree on Computer Science and Automation Engineering in 2013, both at Roma Tre University, in Rome, Italy. Currently, he is employed as Unmanned Vehicle Engineer in the Research and Innovation Department at Info Solution S.p.A.

Gaurav S. Sukhatme is a Professor of Computer Science (joint appointment in Electrical Engineering) at the University of Southern California (USC). He received his undergraduate education at IIT Bombay in Computer Science and Engineering, and M.S. and Ph.D. degrees in Computer Science from USC. He is the co-director of the USC Robotics Research Laboratory and the director of the USC Robotic Embedded Systems Laboratory which he founded in 2000. His research interests are in robot networks with applications to environmental monitoring. He has published extensively in these and related areas. Sukhatme has served as PI on numerous NSF, DARPA and NASA grants. He is a Co-PI on the Center for Embedded Networked Sensing (CENS), an NSF Science and Technology Center. He is a fellow of the IEEE and a recipient of the NSF CAREER award and the Okawa foundation research award. He is one of the founders of the Robotics: Science and Systems conference. He was program chair of the 2008 IEEE International Conference on Robotics and Automation and is program chair of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. He is the Editor-in-Chief of Autonomous Robots and has served as Associate Editor of the IEEE Transactions on Robotics and Automation, the IEEE Transactions on Mobile Computing, and on the editorial board of IEEE Pervasive Computing.