Mobile Netw Appl (2009) 14:322–335
DOI 10.1007/s11036-009-0153-x

Automatic Generation of Persistent Formations for Multi-agent Networks Under Range Constraints

Brian S. Smith · Magnus Egerstedt · Ayanna Howard

Published online: 28 January 2009

Abstract In this paper we present a collection of graph-based methods for determining if a team of mobile robots, subjected to sensor and communication range constraints, can *persistently* achieve a specified formation. What we mean by this is that the formation, once achieved, will be preserved by the direct maintenance of the smallest subset of all possible pairwise inter-agent distances. In this context, formations are defined by sets of points separated by distances corresponding to desired inter-agent distances. Further, we provide graph operations to describe agent interactions that implement a given formation, as well as an algorithm that, given a persistent formation, automatically generates a sequence of such operations. Experimental results are presented that illustrate the operation of the proposed methods on real robot platforms.

Keywords multi-agent network · formations · formation control · graph-based control · decentralized control · persistent graphs · rigid graphs · graph operations

B. S. Smith (✉) · M. Egerstedt · A. Howard
School of Electrical and Computer Engineering,
Georgia Institute of Technology, Atlanta, GA, USA
e-mail: brian@ece.gatech.edu

M. Egerstedt
e-mail: magnus@ece.gatech.edu

A. Howard
e-mail: ayanna.howard@ece.gatech.edu

1 Introduction

Due to recent developments in mobile sensing, computation, and actuation, formation control for multi-agent networks has received significant attention during the last decade. (For example, see [1–14] for a recent, representative sample.) In fact, recent research suggests the use of graph-theoretic structures to represent formations, where vertices represent agents, and edges represent specific inter-agent distances to be maintained through decentralized control laws (see [15–26]).

This work is part of a National Aeronautics and Space Administration (NASA) project to implement a multiple-robot system for research in Antarctica. In this project, a team of geologists at NASA should be able to use a mobile sensor network composed of mobile robots to take sensor readings across ice shelves in order to better understand the impacts of global climate change on the ice shelves. According to specifications, the network should be able to automatically deploy and distribute itself across an area of interest with a user-defined resolution, and to achieve specific, user-defined geometric relationships among the members of the network. The pre-Antarctic stages of this project will be implemented with a prototype multiple-robot system, shown in Fig. 1.

Based on the NASA project as a motivating application, we will study graph-based abstractions of formations, and we define a *target formation* as a set of pairwise, desired inter-agent distances associated with a complete graph, i.e., these distances are specified with respect to all pairs of agents. However, it may not be the case that all inter-agent distances are needed, which leads to the study of so-called *persistent formations* [27].

Fig. 1 The multi-robot network. This network is used in the pre-Antarctic stages of this project. Its mobility and sensor abilities approximate the Antarctic network sufficiently to assemble and deploy formations using the same automatic tools

In a persistent formation, each agent is assigned a set of *constraints*, which are specific inter-agent distances to maintain. These constraints are oriented in the sense that each constraint is the responsibility of a single agent rather than two agents. Maintaining a formation may not require all inter-agent distances to be *directly* maintained by the control laws. Persistent formations typically involve only a proper subset of all possible inter-agent constraints. In [28], graph operations are proposed that, through successive applications, produce a graph corresponding to a persistent formation.

We want to use such operations in order to build persistent formations in the presence of constraints on the effective communication and sensing distances. In fact, these types of constraints were not considered in [28], and the main contribution of this paper is the sequential construction of persistent formations that respect the inter-agent range constraints.

The outline of this paper is as follows: In Section 2, we recall some of the basic definitions needed to set up the problem, followed by a method for determining if a specific target formation is *rigidly feasible* with respect to the range constraints, in Section 3. Next, in Section 4, we show how the same method for determining rigid feasibility also determines *persistent feasibility*. We also present a set of graph operations for building persistent formations with respect to the range constraints, as well as an algorithm for automatically generating a sequence of such operations given a persistent formation graph (Section 5). The experimental results are discussed in Section 6, followed by the conclusions in Section 7.

2 Preliminaries

In this section, we review some of the basic assumptions and terminology needed for the development in later sections. The main object of interest is a persistent formation in which individual robots are responsible for maintaining specific inter-agent distances. Qualitatively, we say that a formation is persistent if, provided that all agents ensure that the distance constraints they are responsible for are satisfied, then the formation is preserved in the sense that *all* pairwise distances are preserved [27].

There are two problems addressed in this paper:

1. Determine if a target formation is persistently feasible given the maximum sensing and communication range of the agents.
2. If the formation is persistently feasible, generate a constraint topology for implementing the formation.

2.1 Network trajectories

We assume that the multi-agent network consists of n agents indexed by $N = \{1, \ldots, n\}$ such that $i \in N$ is the index of agent i. We define $T = [0, \infty)$ as the time interval over which the system is defined. $\forall i \in N$, we define a state $x_i : T \mapsto \mathbb{R}^2$ such that $x_i(t)$ represents the position of agent i at time $t \in T$. We represent the *trajectory* of the network as $X : T \mapsto \mathbb{R}^{2n}$ such that $X(t) = \left[x_1(t)^T, \ldots, x_n(t)^T\right]^T$. We further assume that, $\forall i \in N$, x_i is continuously differentiable with respect to time and, as such, so is X.

2.2 Proximity range Δ

To characterize the sensing and communication abilities of the agents, a *proximity range* $\Delta \in \mathbb{R}^+$ is defined, within which agents can sense and communicate with each other. We assume that any pairs of agents $(i, j) \in N \times N$ can *directly* sense and communicate with each other at time t if and only if they are within Δ of each other, i.e. $\|x_i(t) - x_j(t)\| \leq \Delta$.

2.3 Target formations

We assume that the desired inter-agent distances are defined by a set of n given, relative *positions* $p_i \in \mathbb{R}^2$ $\forall i \in N$ such that $\|p_i - p_j\|$ describes the desired distances between all pairs of agents $(i, j) \in N \times N$. These positions define a *target formation* $P \in \mathbb{R}^{2n}$ such that $P = \left[p_1^T, \ldots, p_n^T\right]^T$.

2.4 Network graph

For the network of size n, we define graph $G_n = (V, E)$ such that

- $V = \{v_1, \ldots, v_n\}$ is the vertex set, and
- $E \subset V \times V$ is the edge set, where each edge in $(v_i, v_j) \in E$ is an *ordered-pair* of vertices such that $v_i \neq v_j$.

In this paper, we utilize a notation such that, for a graph $G = (V, E)$, $V(G) = V$ and $E(G) = E$.

The network graph and a target formation P defines a weight function $\delta : E(G_n) \mapsto \mathbb{R}^+$. Here, δ assigns to each edge $(v_i, v_j) \in E(G_n)$ the desired *length*, or *distance*, between the corresponding points defining P such that $\delta(v_i, v_j) = \|p_i - p_j\|$.

The network graph G_n can be thus be thought of as representing a set of constraints associated with a subset of all pairs of agents, which is why we will refer to this as the *constraint topology* of the network. Also, each edge $(v_i, v_j) \in E(G_n)$ implies that agent i must maintain a constant distance $\|p_j - p_i\|$ from agent j. As such, each edge in G_n models a *constraint* of the network. The direction of the edge implies which agent has the constraint. For example, $\exists (v_i, v_j) \in E(G_n)$ implies that the control law of agent i depends on agent j.

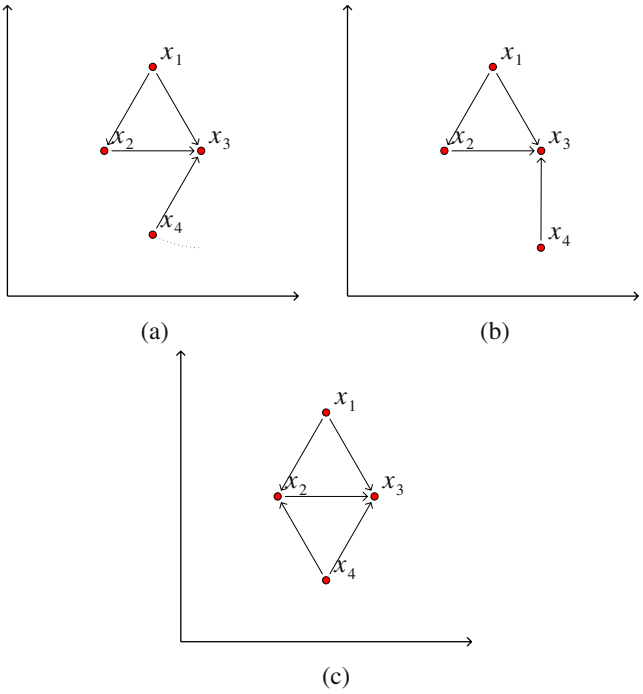


Fig. 2 Rigid and flexible networks. **a** A flexible network. The *dotted line* represents circular motion that agent 4 can perform and still satisfy its constraint with agent 3. **b** Agent 4 can move in a manner that changes its distance to agents 1 and 2. **c** A rigid network. If all constraints are satisfied during continuous motion, then the formation does not change

3 Rigid feasibility and rigid graph generation: the modified "pebble game"

In this section, we present rigid feasibility in terms of range constraints. First, we present rigidity as it has been defined in previous work. Then, we define rigid feasibility under range constraints and provide an algorithm for determining if a given target formation P is rigidly feasible.

3.1 Rigidity

A trajectory X represents the *continuous motion* of a multi-agent network. For a given target formation P, we define a *edge-consistent trajectory* as one such that $\|x_i(t) - x_j(t)\| = \|p_i - p_j\| \ \forall ((v_i, v_j), t) \in E(G_n) \times T$. We define a *rigid trajectory* as one such that $\|x_i(t) - x_j(t)\| = \|p_i - p_j\| \ \forall (i, j, t) \in N \times N \times T$, i.e. the network stays in formation during the trajectory. Thus, a rigid trajectory represents a *rigid motion* of the network. For a given formation P and graph G_n, the multi-agent network is *rigid* if and only if all edge-consistent trajectories of the network are also rigid trajectories. If a network is not rigid, we say that it is *flexible*.

The rigidity of the network for a target formation P and network graph G_n implies that the target formation can be maintained by guaranteeing that the constraints represented by $E(G_n)$ are maintained. Figure 2 gives examples of rigid and flexible multi-agent networks.

3.2 Infinitesimal rigidity

Here, we review the concept of infinitesimal rigidity as presented in [29–31]. The infinitesimal rigidity of a network is a stronger condition than rigidity in that all infinitesimally rigid networks are rigid. While some rigid networks are *not* infinitesimally rigid, the infinitesimal rigidity of a network is a much easier condition to both test for and guarantee through our choice in the topology of G_n and target formation P.

We assume that $x_i(t)$ is continuously differentiable $\forall i \in N$. Since we have defined an edge-consistent trajectory such that the distance between points $x_i(t)$ and $x_j(t)$ remains constant all along the trajectory, this implies that, $\forall \left((v_i, v_j), t\right) \in E(G_n) \times T$,

$$\left(x_i(t) - x_j(t)\right)^T \left(\dot{x}_i(t) - \dot{x}_j(t)\right) = 0. \tag{1}$$

To define a method of predicting infinitesimal rigidity, we (temporarily) assume that $x_i(0) = p_i \ \forall i \in N$. Under this assumption, the assignment of constant instantaneous velocities $u_i \in \mathbb{R}^2 \ \forall i \in N$ such that, $\forall i \in N$, $\dot{x}_i(0) = u_i$ satisfies Eq. 1 $\forall (v_i, v_j) \in E(G_n)$ is described as an *infinitesimal motion* of the network [31]. Let $U \in \mathbb{R}^{2n}$ be defined by the infinitesimal motion such that $U = \left[u_1^T, \ldots, u_n^T\right]^T$. Then Eq. 1 is represented in matrix form $\forall (v_i, v_j) \in E(G_n)$ as

$$M(P, G_n)U = 0,$$

where $M(P, G_n)$ is known as the *rigidity matrix* [31]. The rigidity matrix has $|E(G_n)|$ rows and $2n$ columns. For each edge $(v_i, v_j) \in E(G_n)$, each row m_{ij} of $M(P, G_n)$ represents the equation for that edge as a 2n-vector of the form

$$m_{ij} = (0, \ldots, (p_i - p_j)^T, 0, \ldots, 0, (p_j - p_i)^T, \ldots, 0). \tag{2}$$

Here, $(p_i - p_j)^T$ is in two columns for vertex i, $(p_j - p_i)^T$ is in the columns for j, and zeroes are elsewhere [31]. A network with $n \geq 2$ points in $\mathbb{R}^2$ and a formation P is *infinitesimally rigid* if and only if $rank(M(P, G_n)) = 2n - 3$ [30, 31].

Infinitesimal rigidity implies rigidity, but rigidity does not imply infinitesimal rigidity [29]. Still, the rigidity matrix is an effective way to demonstrate infinitesimal rigidity, and thus rigidity, based on the formation and topology of the network.

3.3 Generic rigidity

It is clear that the rigidity of a network depends both on the topology and the formation. A *generically rigid graph* is an network graph for which there exists a formation P such that the network is infinitesimally rigid. Note that generic rigidity is a property of a network graph, not a network graph and a formation, as is infinitesimal rigidity. Therefore, we refer to generically rigid graphs as *rigid graphs* without confusion.

If G_n is rigid, and the network is infinitesimally rigid with formation $P \in \mathbb{R}^{2n}$, we say that P is a *generic formation* of G_n. If G_n is rigid, then the generic formations of G_n form a dense, open subset of $\mathbb{R}^{2n}$ [30]. This implies that, for any generically rigid graph G_n, any formation P' can be well-approximated by a generic formation P such that the corresponding network is infinitesimally rigid and, therefore, rigid.

3.4 Rigid feasibility

We define *rigid feasibility* as follows:

Definition 1 A target formation P is *rigidly feasible* for a multi-agent network with proximity range Δ if and only if there exists a network graph G_n^Δ such that G_n^Δ is rigid, and $\delta(e) \leq \Delta \ \forall e \in E(G_n^\Delta)$.

It is clear that adding edges to a rigid graph cannot affect its rigidity. A *minimally rigid graph* is rigid but does not remain rigid after the removal of a single edge. By Laman's theorem [32], a network with agents defined in $\mathbb{R}^2$ with $n \geq 2$ vertices is minimally rigid if and only if

1. it has $2n - 3$ edges, and
2. each induced subgraph of $n' \leq n$ vertices has no more than $2n' - 3$ edges.

To generate minimally rigid graphs, we utilize the *"pebble game" algorithm* [33], which is an algorithm for constructing minimally rigid graphs, with a worst case performance of $O(n^2)$ [33]. In the pebble game, each vertex is represented as having two pebbles, each pebble representing a degree of freedom for that vertex. A *pebble covering* exists if each edge can be covered by a pebble from a vertex incident to that edge. To keep track of pebbles, the pebble game works with a directed graph, where a directed edge (v_i, v_j) indicates that edge (v_i, v_j) is covered by a pebble from vertex v_i. For a given $v \in V$, the pebbles of v can only cover edges incident to v.

The pebble game starts with a directed graph with no edges and attempts to add each potential edge one at a time to the pebble covering in a manner that ensures the second part of Laman's theorem is satisfied. Since the pebbles of each vertex limit the number of edges directed out of each vertex, this is accomplished by modifying the directions of both the edge to be added and the other edges already in the graph. If part 2 of Laman's theorem is satisfied, we say that a *valid pebble covering* has been found. If a valid pebble covering of $2n - 3$ such edges is found, then this implies that the first part of Laman's theorem is satisfied and the graph is minimally rigid. For more detail on the implementation of this algorithm, see [33].

To test for a minimally rigid graph that satisfies Definition 1, we modify the pebble game algorithm so that it only considers edges of length less than or equal to Δ. The modified pebble game is described in Algorithm 1.

Algorithm 1 *Modified Pebble Game*(P, Δ)

Require: P is a formation of n points
  Initialize G_n^Δ such that $V(G_n^\Delta) := \{v_1, \ldots, v_n\}$, $E(G_n^\Delta) := \emptyset$
  Initialize *rigid* := **false**
  for all possible edges $e = (v_i, v_j) \in V(G_n^\Delta) \times V(G_n^\Delta)$ such that $\delta(e) \leq \Delta$ and **while** *rigid* = **false do**
    $E(G_n^\Delta) := E(G_n^\Delta) \cup e$;
    Rearrange edge directions to try to find a valid pebble covering;
    if a valid pebble covering is *not* found **then**
      $E(G_n^\Delta) := E(G_n^\Delta) \setminus e$;
    end if
    if $|E(G_n^\Delta)| = 2n - 3$ **then**
      rigid := **true**;
    end if
  end for
  return (*rigid*, G_n^Δ);

The following Theorem 1 states the effectiveness of the modified pebble game to test for rigid feasibility.

Theorem 1 *A target formation P is rigidly feasible for a multi-agent network with proximity range Δ if and only if the algorithm Modified Pebble Game(P, Δ) returns a minimally rigid graph.*

Proof Definition 1 is satisfied for formation P only if there exists a rigid graph G_n^Δ such that $\delta(e) \leq \Delta$ $\forall e \in E(G_n^\Delta)$. This implies the existence of a minimally rigid graph with the same properties. Assume that G_n^Δ exists, but that *Modified Pebble Game*(P, Δ) fails to return a minimally rigid graph. Note from [33] that the unmodified pebble game generates a rigid graph by considering each edge and adding it to a flexible graph until it becomes minimally rigid. Therefore, the failure of *ModifiedPebbleGame*(P, Δ) implies that no such graph can be generated considering only edges such that their distance in the network would be less than or equal to Δ. Since the unmodified pebble game always returns a minimally rigid graph [33], this implies that all rigid graphs result in a network with $\delta(e) > \Delta$ for some $e \in E(G_n^\Delta)$. However, this violates our assumption that G_n^Δ exists. Therefore, the formation is rigidly feasible only if *Modified Pebble Game*(P, Δ) returns a minimally rigid graph.

If the modified pebble game produces such a minimally rigid graph such that $\delta(e) \leq \Delta$ $\forall e \in E(G_n^\Delta)$, then the conditions of Definition 1 are satisfied. □

4 Persistent feasibility and persistent graph generation

In this section, we present persistent feasibility in terms of range constraints. First, we present persistence as it has been defined in previous work. Then, we define persistent feasibility under range constraints. Further, we demonstrate that rigid feasibility and persistent feasibility are equivalent. We also show that the modified pebble game algorithm generates minimally persistent graphs.

4.1 Persistence

Persistence is a quality of networks that is very closely related to the concept of *constraint consistence*. Informally, we say that constraint consistence means that all constraints are satisfied as long as all agents satisfy their individual constraints, i.e., no subset of agents can satisfy their constraints in a manner which prevents another agent from satisfying a constraint. Constraint consistence is determined by the number and orientation of the constraints. Figure 3 shows constraint consistent and inconsistent networks. For a more rigorous

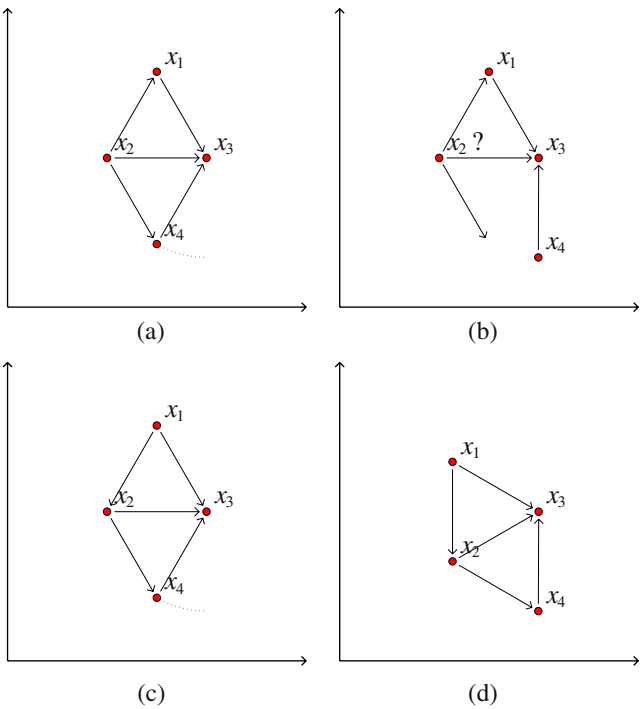


Fig. 3 Persistence example. **a** A network that is not persistent. Here, agent 4 can perform circular motion around agent 3. **b** If agent 4 moves, agent 2 cannot move in a way that preserves the distances between agent 2 and agents 1, 3, and 4. **c** A persistent network. It is constraint consistent and rigid. Agents 3 and 4 are a leader–follower pair. **d** If agent 4 satisfies its constraint, the other agents maintain formation during continuous motion

definition, see [27]. A network is persistent if and only if it is rigid and constraint consistent [27]. Figure 3c and d show a persistent network.

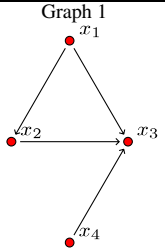
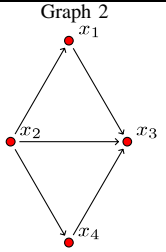
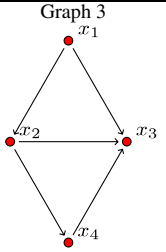
We say that a network graph G_n is *generically constraint consistent* if all of its vertices have an out-degree less than or equal to 2 [27]. This applies to graphs as well. Thus, we refer to generically constraint consistent graphs as being *constraint consistent graphs*.

Similar to generic rigidity, we say that an network graph G_n is *generically persistent* if it is generically rigid and generically constraint consistent. Like generic rigidity, generic persistence applies to graphs, not networks. Therefore, we refer to generically persistent graphs as persistent graphs without confusion. A persistent graph is *minimally persistent* if it is persistent and if no edge can be removed without losing persistence [27].

4.2 Rigidity, constraint consistence, and persistence summary

To summarize the topological notions of rigidity, constraint consistence, and persistence and their relations, see Table 1. Rigidity tells us whether or not we have sufficient edges in our graph to guarantee that the formation is maintained by only maintaining its edge lengths. Therefore, in Table 1, Graph 1 is flexible (i.e. not rigid), while Graphs 2 and 3 are rigid. Note that rigidity does not depend on the orientation of the edges. Unlike rigidity, constraint consistence *does* depend on the orientation of the edges. While Graph 1 is flexible, it is constraint consistent, since all vertices have an out-degree less than or equal to 2. This implies that they can maintain these edges regardless of how agents 3 and 4 move while respecting the constraint agent 4 has with agent 3. Still, the formation may deform, since the graph is flexible. On the other hand, Graph 2 is rigid, but not constraint consistent. While the maintenance of the edges would preserve the formation, it is possible for agent 4 to move such that agent 2 cannot maintain all of these edges, as in Fig. 3. Of the graphs in Table 1, only Graph 3 is both rigid and constraint consistent, which makes it the only persistent graph example. The maintenance of the edges ensures that the formation does not deform, and all agents can, in fact, maintain these edges during any edge consistent motion.

Table 1 Rigidity, constraint consistence, and persistence examples table

Graph 1	Graph 2	Graph 3
		
Flexible	Rigid	
Constraint Consistent	Constraint Inconsistent	Constraint Consistent
Not Persistent	Not Persistent	*Persistent*

4.3 Persistent feasibility

We define *persistent feasibility* as follows:

Definition 2 A target formation defined by configuration P is *persistently feasible* for a multi-agent network with proximity range Δ if and only if a exists an network graph G_n^Δ such that G_n^Δ is persistent, and $\delta(e) \leq \Delta \ \forall e \in E(G_n^\Delta)$.

For any minimally rigid graph, it is possible to assign directions to the edges such that the obtained directed graph is minimally persistent [28]. Therefore, we have the following Theorem 2 describing necessary and sufficient conditions for a target formation to be persistently feasible.

Theorem 2 *For a multi-agent network with proximity range Δ, a target formation P is persistently feasible if and only if it is rigidly feasible.*

Proof If P is rigidly feasible, then, by Definition 1, there exists a minimally rigid graph G_n^Δ such that the network is rigid and $\delta(e) \leq \Delta \ \forall e \in E(G_n^\Delta)$. This implies that the directions of the edges of G_n^Δ can be assigned such that it is a persistent graph, implying that rigidly feasible formations are persistently feasible. Since a network is persistent if and only if it is rigid and constraint consistent, then P is not persistently feasible if it is not rigidly feasible. □

Theorem 2 shows that the modified pebble game tests for both rigid and persistent feasibility.

4.4 Persistent graph generation

Here, we show that the pebble game algorithm also generates minimally persistent graphs.

A graph is minimally persistent if and only if it is minimally rigid and no vertex has an out-degree larger than two [27]. We denote the out-degree of a vertex v by $deg^-(v)$. Note that the pebble game produces a directed graph G_n^Δ, where each edge (v_i, v_j) is covered

by one of two pebbles from vertex v_i. Thus, we have the following theorem:

Theorem 3 *The pebble game and modified pebble game algorithms generate minimally persistent graphs.*

Proof Assume that G_n^Δ is a rigid graph successfully generated by the pebble game. In [33], it is shown that the pebble game generates a minimally rigid graph. Since each directed edge $(v_i, v_j) \in E(G_n^\Delta)$ represents the edge being covered by a pebble from vertex v_i, this implies that $deg^-(v_i) \leq 2\ \forall v_i \in V(G_n^\Delta)$. This implies that G_n^Δ is minimally persistent. This also holds for the modified pebble game. □

5 Graph operations

In this section, we describe methods for representing and choosing leader–follower pairs of a persistent formation. We present graph operations that represent agent interactions that execute a persistent formation. We also present an algorithm for generating a sequence of graph operations, which represents a sequence of agent interactions to execute a persistent formation.

5.1 Leader–follower pairs

We define a *leader–follower pair* [16] as a pair of adjacent vertices $(v_l, v_f) \in V(G_n^\Delta) \times V(G_n^\Delta)$ such that $deg^-(v_l) = 0$, $deg^-(v_f) = 1$, and $\exists (v_f, v_l) \in E(G_n^\Delta)$. We say that vertex v_l is the *leader vertex*, and vertex v_f is the *follower vertex*.

The leader agent has no constraints, and thus has two degrees of freedom, implying that the persistent formation will follow the leader agent in $\mathbb{R}^2$. Similarly, the follower agent has one constraint, and thus one degree of freedom, implying that the persistent formation will rotate around the leader agent as the follower agent performs circular motion around the leader. For a persistent graph, edge-reversing operations can make any pair of adjacent agents a leader–follower pair with the graph remaining persistent [28]. A leader–follower pair is demonstrated in Fig. 3c and d.

5.2 Persistent graph operations

In an actual multi-agent network, achieving a persistent formation requires agents with no constraints to interact and establish constraints. Such a sequence of agent interactions, if successful, results in a persistent formation, with inter-agent distances corresponding to the target formation.

Graph operations can be used to represent such a sequence of agent interactions. In [28], graph operations are presented for assembling and modifying persistent graphs. These operations consist of directed vertex addition and edge-splitting operations. Consider a graph G such that $\{v_i, v_j, v_p\} \subset V(G)$, $(v_p, v_j) \in E(G)$, and $v_k \notin V(G)$. A vertex addition consists of adding v_k to $V(G)$ and adding edges (v_k, v_i), (v_k, v_j) to $E(G)$. Figure 4a and b show a vertex addition operation. An edge-splitting operation consists of adding v_k to $V(G)$ and adding edges (v_k, v_i), (v_k, v_j) to $E(G)$, while also removing edge (v_p, v_j) from $E(G)$. Figure 4c and d show an edge-splitting operation. Graph operation sequences for assembling minimally persistent graphs are typically generated by performing inverse graph operations on the graph to be assembled, along with edge reversing operations.

In [28], it is shown that any persistent graph can be deconstructed by a combination of these inverse operations, and then reconstructed by a reverse sequence of

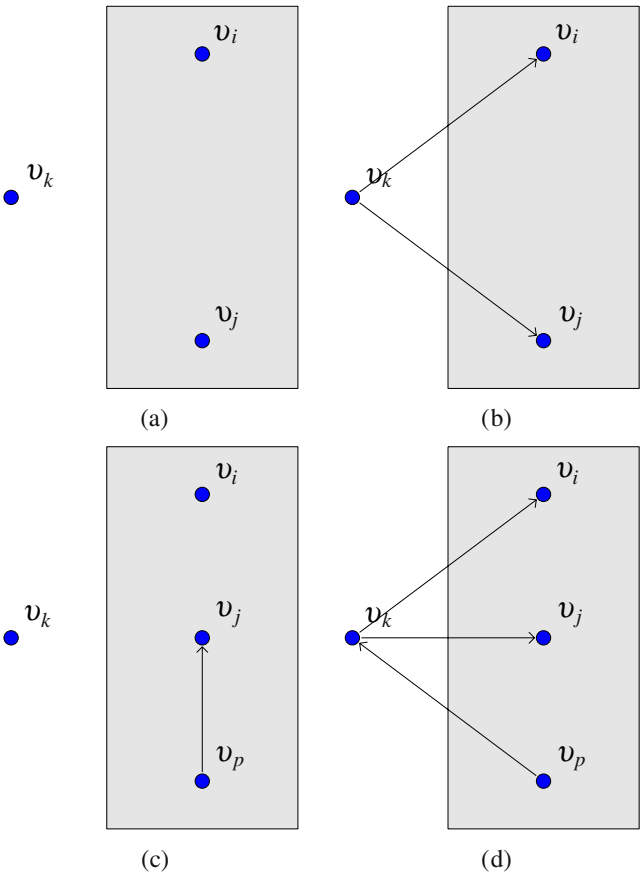


Fig. 4 Persistent graph operations. **a** and **b** show the results of a vertex addition operation. **c** and **d** show the results of an edge-splitting operation. In this figure, the *shaded area* represents a minimally persistent graph before the operation. The resulting graph is always minimally persistent, as well

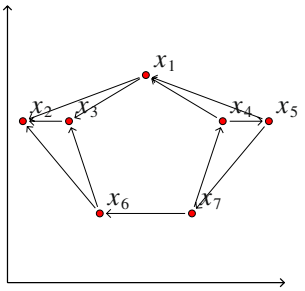


Fig. 5 An example network where performing an inverse edge-splitting operation introduces a new edge whose length is greater than all pre-existing edges. This new edge could violate the proximity range of the network

non-inverse operations. Additionally, [28] guarantees that each intermediate graph is persistent. However, these methods are completely graph based, and do not take into account a proximity range for a multi-agent network. Consider Fig. 5. This network has a minimally persistent graph. An inverse vertex addition cannot be performed. Also, note that any inverse edge-splitting operation will introduce a new edge into the network which has a length longer than any other edge. This new edge could violate the proximity range of the mobile agent network. *Therefore, given a formation and a proximity range limit on the edge lengths of a network, certain network graphs cannot be deconstructed by these traditional operations without introducing a constraint that violates the proximity range.*

5.3 Persistent-Δ operations

In this section we present two new graph operations to construct persistent graphs. These, combined with traditional vertex addition, allow any persistent graph with a leader–follower pair to be constructed without using any edges that are not contained in the final graph. We call this set of three graph operations *persistent-Δ operations*.

Each operation is represented by a double $op = (V, E)$, where $V(op) = V$ is a set of vertices to add to the graph, and $E(op) = E$ is a set of edges to add to the graph.

A *vertex addition* is a persistent-Δ operation defined as in Section 5.2. A vertex addition is represented as $vertexAddition(v_i, v_j, v_k) = (\{v_k\}, \{(v_k, v_i), (v_k, v_j)\})$.

Consider a directed graph G such that $v_i \in V(G)$, $v_j \notin V(G)$. *Single-vertex addition* consists of adding a vertex v_j to $V(G)$ and adding edge (v_j, v_i) to $E(G)$. A single-vertex addition is represented as $singleVertex(v_i, v_j) = (\{v_j\}, \{(v_j, v_i)\})$. Note that this operation does *not* preserve persistence. In fact, it guarantees a loss of persistence, since this new vertex has one degree of freedom.

Consider a directed graph G such that $(v_i, v_j) \in V(G) \times V(G)$ and $(v_j, v_i) \notin E(G)$. *Edge insertion* consists of adding edge (v_j, v_i) to $E(G)$. An edge insertion is represented as $edgeInsertion(v_i, v_j) = (\emptyset, \{(v_j, v_i)\})$. Figure 6 shows these operations.

5.4 Persistent-Δ sequence generation

This section describes how persistent-Δ operations can be used to construct any persistent graph with a leader–follower pair.

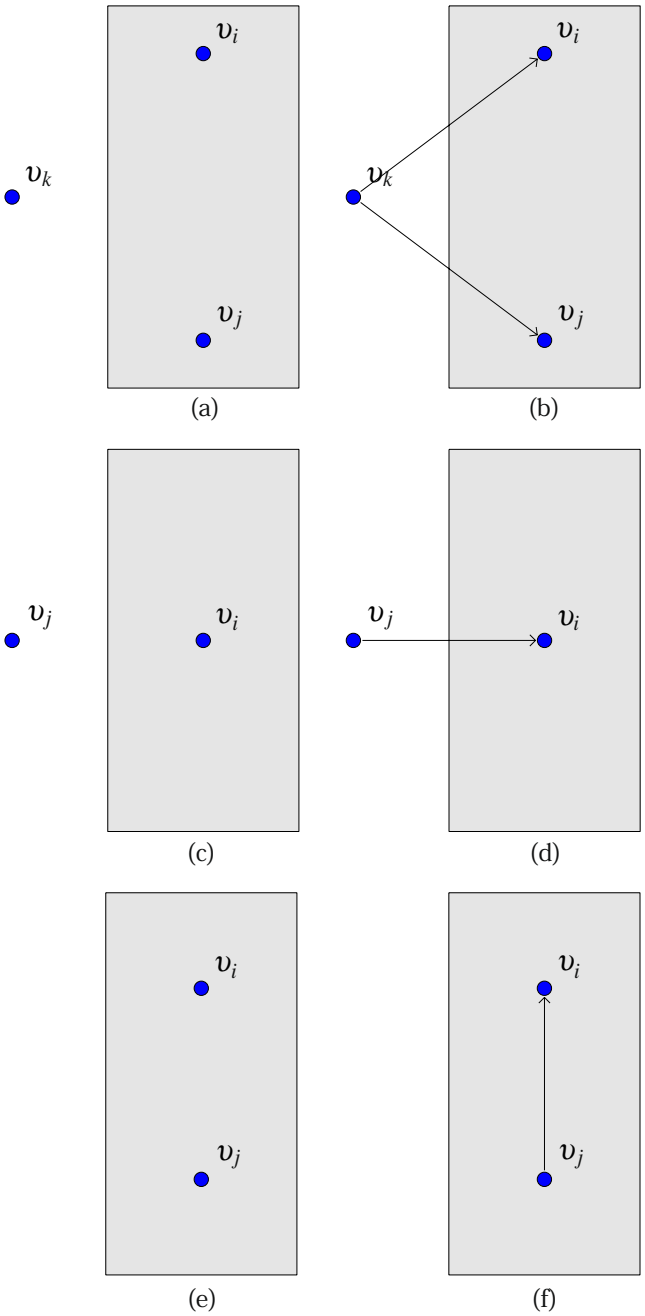


Fig. 6 Persistent-Δ graph operations. **a**, **b** a vertex addition. **c**, **d** a single-vertex addition. **e**, **f** an edge insertion operation. As before, the *shaded area* represents a minimally persistent graph before the operation

If G_n^Δ is a minimally persistent graph where $\exists(v_i, v_j) \in V(G_n^\Delta) \times V(G_n^\Delta)$ such that $deg^-(v_i) \geq 1$ and vertex $deg^-(v_j) \leq 1$, then there is a directed path from v_i to v_j [28]. Also, if $(v_l, v_f) \in V(G_n^\Delta) \times V(G_n^\Delta)$ are a leader–follower pair, respectively, then, for all vertices $v \in V(G_n^\Delta) \setminus \{v_l, v_f\}$, $deg^-(v) = 2$ [27]. This leads to the following lemma:

Lemma 1 *Let G_n^Δ be a minimally persistent graph such that vertex v_l is the leader vertex and vertex v_f is the follower vertex of a leader–follower pair. This implies the existence of a directed path from all vertices $v \in V(G_n^\Delta) \setminus v_l$ to v_l.*

Proof Assume that G_n^Δ exists as in Lemma 1. Since v_l and v_f are a leader–follower pair, this implies a directed path from v_f to v_l and that $deg^-(v_l) < deg^-(v_f) \leq 1$. This implies that all vertices $v \in V(G_n^\Delta) \setminus \{v_l, v_f\}$, $deg^-(v) = 2$. Then there is a directed path from v to v_f and v_l. This implies that there exists path from all vertices in $V(G_n^\Delta) \setminus v_l$ to v_l. □

This leads us to an algorithm for constructing a sequence of graph operations to construct a minimally persistent graph. We define a *leader–follower seed* as a graph G_2 such that $V(G_2) = \{v_l, v_f\}$ and $E(G_2) = \{(v_f, v_l)\}$. Here, vertex v_l is the leader vertex, and vertex v_f is the follower vertex.

Any minimally persistent graph can be constructed from a leader–follower seed by a sequence of

Algorithm 2 *Persistent*Δ *Generation*(G_n)

Require: Graph G_n^Δ exists such that G_n^Δ is minimally persistent with leader–follower pair (v_l, v_f).
  Initialize leader–follower seed $G := G_2$ such that $V(G) := V(G_2) = \{v_l, v_f\}$ and $E(G) := E(G_2) = \{(v_f, v_l)\}$;
  Initialize sequence of graph operations $S := \emptyset$;
  while $|V(G)| < |V(G_n^\Delta)|$ or $|E(G)| < |E(G_n^\Delta)|$ **do**
    Initialize sequence of graph operations $s := \emptyset$;
    for all $(v_i, v_j) \in V(G) \times V(G)$ **do**
      {Generate all possible edge insertions}
      if $(v_j, v_i) \in E(G_n^\Delta)$ and $(v_j, v_i) \notin E(G)$ **then**
        $ei := edgeInsertion(v_i, v_j)$;
        $s := s \cdot ei$;
      end if
    end for
    Initialize $vertexAdded :=$ **false**;
    for all $v_k \in V(G_n^\Delta)$ such that $v_k \notin V(G)$ **do**
      {Generate all possible vertex additions}
      if $\exists(v_i, v_j) \in V(G) \times V(G)$ such that $\{(v_k, v_i), (v_k, v_j)\} \in E(G_n^\Delta)$ **then**
        $va := vertexAddition(v_i, v_j, v_k)$
        $s := s \cdot va$;
        $vertexAdded :=$ **true**;
      end if
    end for
    if $vertexAdded =$ **false then**
      for all $v_j \in V(G_n^\Delta)$ such that $v_j \notin V(G)$ **do**
        {Generate all possible single-vertex additions}
        if $\exists v_i \in V(G)$ such that $(v_j, v_i) \in E(G_n^\Delta)$ **then**
          $sva := singleVertex(v_i, v_j)$;
          $s := s \cdot sva$;
        end if
      end for
    end if
    for all operations $op \in s$ **do**
      {Perform all determined graph operations}
      $V(G) := V(G) \cup V(op)$;
      $E(G) := E(G) \cup E(op)$;
    end for
    $S := S \cdot s$;
  end while
  return S;

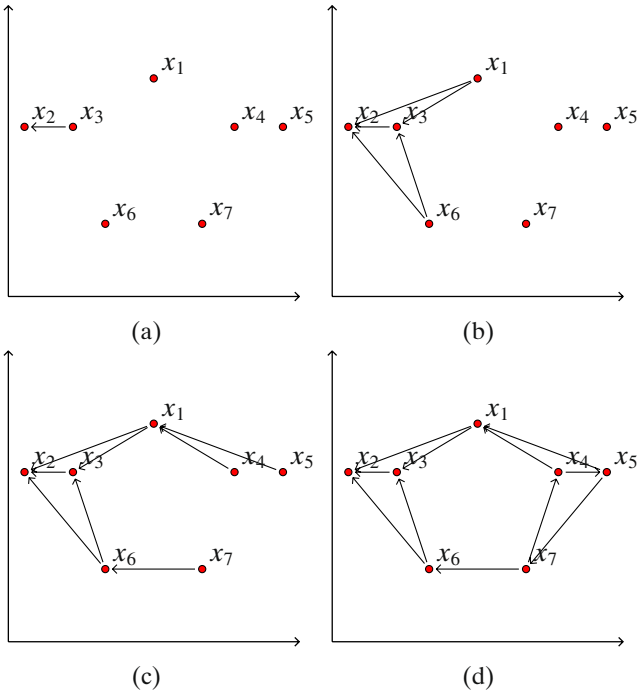


Fig. 7 A sequence of Persistent-Δ operations constructing a framework. **a** The initial leader–follower seed. **b** Two vertex additions are performed. **c** No more vertex additions are possible. Three single-vertex additions are performed. **d** Three edge insertions are performed, one for each single-vertex addition

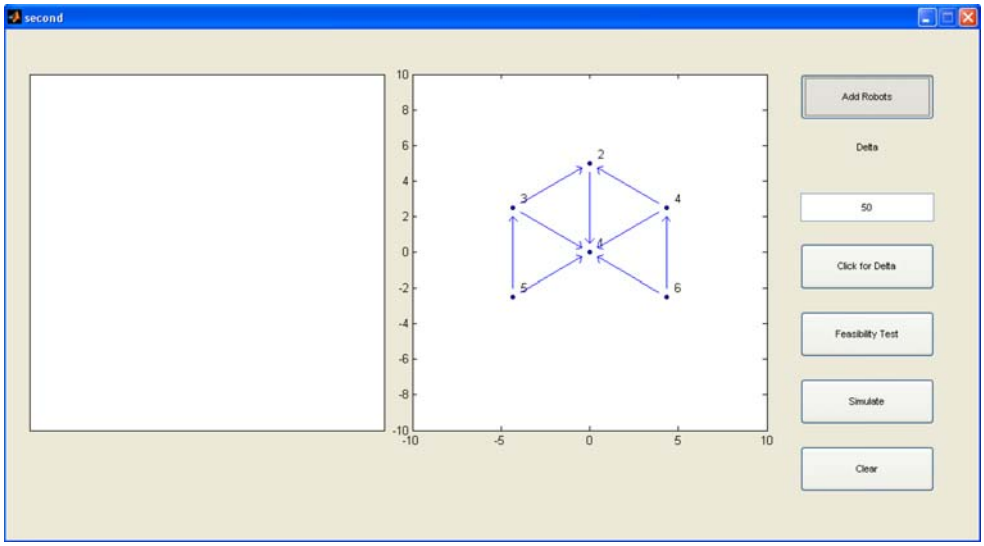


Fig. 8 The Graphical User Interface (GUI) for specifying formations. By defining the formation P and the proximity range Δ, the software uses our methods to determine if the formation is persistently feasible. If so, it automatically generates a sequence of persistent-Δ graph operations for assembling such a formation, as well as analogous rules for an Embedded Graph Grammar (EGG) system to assemble the formation. The EGG rules are used by the network to assemble formations with the multi-robot network in Fig. 10

persistent-Δ graph operations. First, given a minimally persistent graph G_n^Δ, a graph G is initialized to the leader–follower seed G_2 using the leader and follower vertices in G_n^Δ. Until all vertices and edges of G_n^Δ are present in G, the following process is performed:

1. Generate each possible edge insertion.
2. Generate each possible vertex addition.
3. If no vertex additions were performed, generate each possible single-vertex addition.

The condition for single-vertex addition is due to the fact that single-vertex addition does not preserve persistence. Directed vertex addition does. Therefore, these are preferred. Edge insertions are necessary to complete the graph after single-vertex additions are performed. After this process, each of the generated graph operations is executed on the graph G. This process is repeated until all vertices and edges have been added to the graph. Algorithm 2 describes this process. In Algorithm 2, we represent concatenating element s to the end of sequence S by $S \cdot s$.

Figure 7 shows a resulting sequence of this algorithm. We have the following theorem for the effectiveness of this method:

Theorem 4 *For a minimally persistent graph G_n^Δ with a leader–follower pair, the persistent-Δ generation algorithm will generate a sequence of graph operations that construct G_n^Δ from a leader–follower seed.*

Proof Assume that G_n^Δ exists, with (v_l, v_f) as the leader and follower of a leader–follower pair, and that G is the initialized leader–follower seed. If the graph has only two vertices, the graph is constructed.

If there are more than two vertices, then, by Lemma 1, there exists a path from all vertices in $V(G_n^\Delta) \setminus \{v_l\}$ to vertex v_l. This implies that there exists a pair of vertices (v_i, v_j) such that $v_j \in V(G_n^\Delta)$, $v_j \notin V(G)$, $v_i \in V(G)$, $(v_j, v_i) \in E(G_n^\Delta)$. This implies that a single-vertex addition is possible (there may also be vertex additions possible, but this is unnecessary for the proof).

Assume that a single-vertex operation is performed, increasing the size of $V(G)$ and $E(G)$. Note that G always has the leader–follower pair. Therefore, if there are remaining vertices $v \in V(G_n^\Delta)$ such that $v \notin V(G)$, then Lemma 1 also shows that more single-vertex additions are possible. In fact, more single-vertex additions will always be possible until there does not exist a $v \in V(G_n^\Delta)$ such that $v \notin V(G)$. Since we have not added any vertices $v \notin V(G_n^\Delta)$ to $V(G)$, this implies that, at this point, $V(G_n^\Delta) = V(G)$.

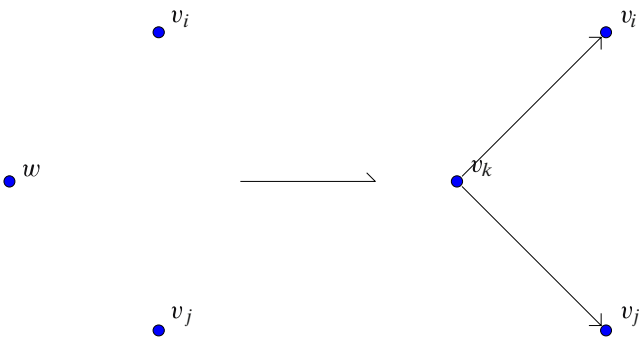


Fig. 9 A vertex addition rule. Here, robots whose labels correspond to v_i and v_j imply that these robots have been assigned to positions i and j in the formation. Each vertex addition graph operation (as depicted in Fig. 6) defines a vertex addition rule as shown in this figure. These rules allow the formation to be assembled

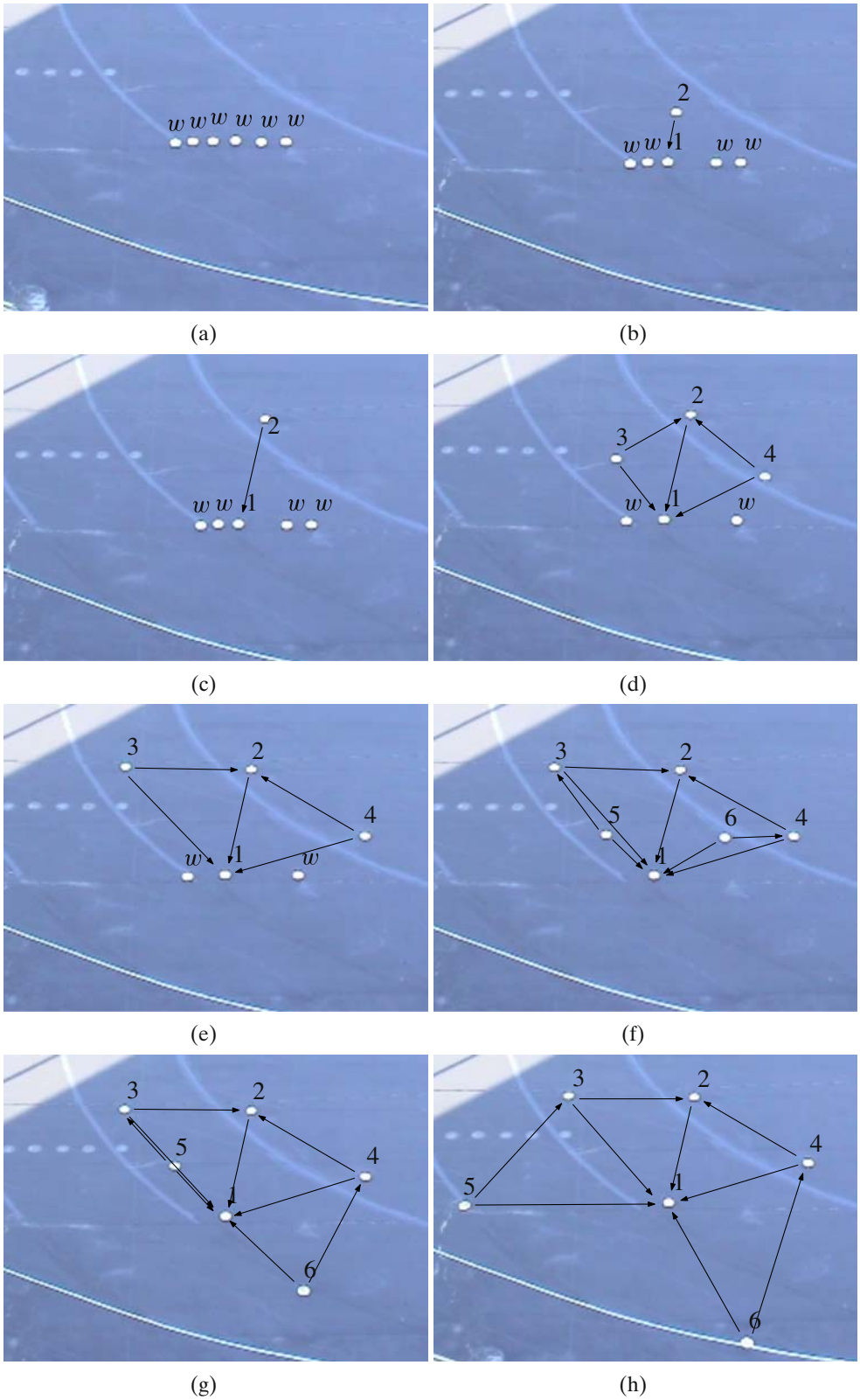


Fig. 10 EGG execution of the graph operations on the multi-robot network. **a** shows each robot labeled as w. **b** shows the results of the leader–follower seed graph being generated. Now a follower (robot 2) begins to satisfy the constraint indicated by the edge to the leader (robot 1, shown in **c**. Once the follower has finalized its position, two vertex additions are executed, shown in **d**. Robots labeled 3 and 4 begin to satisfy their constraints with the robots labeled 1 and 2, as shown in **e**. Similarly, **f** and **g** show two more concurrent vertex additions. **h** shows the completed formation

For all edges $(v_j, v_i) \in E(G_n^{\Delta})$, either $(v_j, v_i) = (v_f, v_l)$,the leader–follower edge, or (v_j, v_i) is not the leader–follower edge. If (v_j, v_i) is the leader–follower edge, then it was added to $E(G)$ when the leader–follower seed was initialized. If it is not the leader–follower edge, note that we have already proven that all

vertices $V(G_n^\Delta)$ are added to $V(G)$ such that $V(G_n^\Delta) = V(G)$. This implies that, for any remaining edges not added by vertex or single-vertex additions, there exists a pair of vertices $(v_i, v_j) \in V(G)$ such that $(v_j, v_i) \in E(G_n^\Delta)$ and $(v_j, v_i) \notin E(G)$. These edges are added by edge insertions.

Since the algorithm uses these conditions to search for single-vertex additions and edge-insertions, all such operations are performed, guaranteeing that $V(G_n^\Delta) = V(G)$ and $E(G_n^\Delta) = E(G)$. □

6 Implementation scenario and results

Here, we demonstrate the assembly of formations on a multi-robot network using graph operations.

In [34], we consider automatic methods for implementing a subset of minimally persistent formations with leader–follower pairs. Specifically, we consider minimally persistent formations that correspond to *stably rigid graphs* [35, 36]. Since stably rigid graphs are acyclic, the implementation of these formations with control laws is simplified. As a consequence, these graphs can be assembled from a leader–follower seed by sequences of vertex additions only.

We implement the following scenario: We have a network of six robots with data collection sensors, and we wish to distribute them in a 5 m triangular coverage pattern over an area of interest. Triangular coverage patterns occur frequently, since they dictate an equal distance (in this case, of 5 m) between each adjacent robot in the coverage pattern. Therefore, we enter a triangulation pattern of positions in our graphical program discussed in Section 2 and shown in Fig. 8.

The points entered in the GUI define our target formation P. The modified pebble game is used to define the minimally persistent graph G_n^Δ shown in Fig. 8, as well as a leader–follower seed G_2 (here, with vertices 1 and 2), and a sequence of vertex addition operations that define a Henneberg sequence S.

To implement these graph operations with the network, we us an *Embedded Graph Grammar (EGG)* system [34]. In this system, a *rule* is defined for each graph operation, as well as the assembly of the leader–follower seed graph. The EGG system deals with labeled graphs where each label corresponds to the position in the formation assigned to each robot. Each rule has a *left graph* L and a *right graph* R, and this pair in rule r is denoted as $(L \rightharpoonup R) \in r$. We define G_n as the network graph corresponding to the actual robot network. When an induced subgraph of G_n is a label-preserving isomorphism of L, it can be replaced by R. Thus, we can define rules that correspond to vertex additions for implementing formation assembly. Figure 9 shows how each vertex addition graph operation (Fig. 6) define vertex addition rules in the EGG system.

Initially, each robot begins with a label of w, indicating that the robot is not assigned a position. First, the leader–follower rule assigns the leader and follower position to two robots. After this, vertex addition rules (corresponding to vertex addition operations) assign other positions to other robots. As such, the topology of G_n changes as the system evolves. Finally, each robot is assigned a position, and the labeling of G_n is an isomorphism from G_n to G_n^Δ, the desired network graph for our target formation P. The EGG system also describes the *mode* of each robot, corresponding to the vertex it is assigned in G_n^Δ. The topology of G_n^Δ and the geometry defined by the target formation P define the control laws for each robot according to the position it is assigned in the formation. This combination of assembly rules and control laws result in the geometry of the assembled formation corresponding to the target formation P, with a topology corresponding to G_n^Δ (For a more detailed explanation of the definition of this EGG system, see [34]).

Figure 10 shows the execution results of the scenario. In this figure, each robot is labeled with either w, indicating that it is a wanderer, or with the number corresponding to its vertex in Fig. 8. In Fig. 10a, we see the initial setup, where each robot is a wanderer. First, a leader–follower pair is formed such that one of the robots is now a leader (labeled 1), and the other is a follower (labeled 2), and the follower begins moving to reach a distance of 5 m from the leader, as shown in Fig. 10b and c.

As shown in Fig. 10d and e, two vertex addition position operations are applied simultaneously (robots labeled 3 and 4). Similarly, Fig. 10f and g show two concurrent vertex addition operations being applied (robots labeled 5 and 6). Finally, the formation is successfully completed, as shown in Fig. 10h.

7 Conclusions

In this paper, we presented a method for determining if given target formations for multi-agent networks are rigidly feasible and persistently feasible in the sense that they can be realized by a team of mobile agents with limited sensing and communication range. We introduced an algorithm for generating minimally persistent graphs under such proximity constraints. We also presented new graph operations to construct a persistent graph that represents a formation under range

constraints, as well as a method for automatically generating a sequence of these operations for any formation in question. These graphs and operations describe the control and coordination strategies necessary to allow the desired formation to emerge in a multi-agent network. Experimental results were given that show that the developed methods can be implemented on a real robot network.

Acknowledgements This work was partially supported under a contract with the National Aeronautics and Space Administration. We also thank Julien Hendrickx for helpful discussions about graph rigidity and persistence.

References

1. Balch T, Arkin RC (1998) Behavior-based formation control for multirobot teams. IEEE Trans Robot Automat 14(6):926–939, Dec
2. Cortés J, Martínez S, Bullo F (2006) Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. IEEE Trans Robot Automat 51(8):1289–1298, Aug
3. Desai J, Ostrowski J, Kumar V (2001) Modeling and control of formations of nonholonomic mobile robots. IEEE Trans Robot Automat 17(6):905–908, Dec
4. Do KD, Pan J (2007) Nonlinear formation control of unicycle-type mobile robots. Robot Auton Syst 55(3):191–204, March
5. Egerstedt M, Hu X (2001) Formation constrained multiagent control. IEEE Trans Robot Automat 17(6):947–951, Dec
6. Kalantar S, Zimmer UR (2007) Distributed shape control of homogeneous swarms of autonomous underwater vehicles. Auton Robots 22(1):37–53, January
7. Kaminka GA, Glick R (2006) Towards robust multi-robot formations. In: Conference on international robotics and automation, Orlando, 15–19 May 2006, pp 582–588
8. Lawton J, Beard R, Young B (2003) A decentralized approach to formation maneuvers. IEEE Trans Robot Automat 19(6):933–941, Dec
9. Leonard NE, Fiorelli E (2001) Virtual leaders, artificial potentials and coordinated control of groups. In: Proceedings of the IEEE conference on decision and control 2001, Orlando, December 2001, pp 2968–2973
10. Lin J, Morse A, Anderson B (2003) The multi-agent rendezvous problem. In: Proceedings of the 42nd IEEE conference on decision and control, Maui, December 2003, pp 1508–1513
11. Sugihara K, Suzuki I (1990) Distributed motion coordination of multiple robots. In: Proceedings of IEEE int. symp. on intelligent control, Philadelphia, 5–7 September 1990, pp 138–143
12. Vig L, Adams JA (2006) Multi-robot coalition formation. IEEE Trans Robot 22(4):637–649, August
13. Yuan L, Weidong C, Yugeng X (2006) Energy-efficient aggregation control for mobile sensor networks. In: International conference on intelligent computing, vol 344. Intelligent control and automation, Kunming, August 2006, pp 188–193
14. Zhijun T, Ozguner U (2006) On non-escape search for a moving target by multiple mobile sensor agents. In: American control conference, American automatic control council, IEEE, Minneapolis, June 2006, p 6
15. Ando H, Oasa Y, Suzuki I, Yamashita M (1999) Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans Robot Automat 15:818–828, Oct
16. Eren T, Whiteley W, Anderson BDO, Morse AS, Belhumeur PN (2005) Information structures to secure control of rigid formations with leader–follower architecture. In: Proceedings of the American control conference, Portland, June 2005, pp 2966–2971
17. Fax JA, Murray RM (2002) Graph laplacian and stabilization of vehicle formations. In: Proceedings of the 15th IFAC conf, Barcelona, 21–26 July 2002, pp 283–288
18. Fax J, Murray R (2004) Information flow and cooperative control of vehicle formations. IEEE Trans Automat Contr 49:1465–1476, Sept
19. Jadbabaie JLA, Morse AS (2003) Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans Automat Contr 48(6):988–1001, June
20. Ji M, Egerstedt M (2007) Distributed coordination control of multi-agent systems while preserving connectedness. IEEE Trans Robot 23:693–703
21. Lin Z, Broucke M, Francis B (2004) Local control strategies for groups of mobile autonomous agents. IEEE Trans Automat Contr 49(4):622–629
22. Kim Y, Mesbahi M (2006) On maximizing the second smallest eigenvalue of a state-dependent graph laplacian. IEEE Trans Automat Contr 51:116–120, Jan
23. Ren W, Beard R (2004) Consensus of information under dynamically changing interaction topologies. In: Proceedings of the American control conference 2004, vol 6, Boston, 30 June–2 July 2004, pp 4939–4944
24. Saber RO, Murray RM (2002) Distributed structural stabilization and tracking for formations of dynamic multiagents. In: Proceedings of the 41st IEEE conference on decision and control 2002, vol 1, Las Vegas, December 2002, pp 209–215
25. Saber RO, Murray RM (2003) Flocking with obstacl avoidance: cooperation with limited communication in mobile networks. In: Proceedings of the 42nd IEEE conference on decision and control 2003, vol 2, Maui, December 2003, pp 2022–2028
26. Saber RO, Murray RM (2003) Agreement problems in networks with directed graphs and switching toplogy. In: Proceedings of the 42nd IEEE conference on decision and control 2003, vol 4, Maui, December 2003, pp 4126–4132
27. Hendrickx JM, Anderson BDO, Delvenne J-C, Blondel VD (2000) Directed graphs for the analysis of rigidity and persistence in autonomous agent systems. Int J Robust Nonlinear Control 17:960–981
28. Hendrickx JM, Fidan B, Yu C, Anderson BDO, Blondel VD (2006) Elementary operations for the reorganization of minimally persistent formations. In: Proceedings of the mathematical theory of networks and systems (MTNS) conference, no. 17, Kyoto, July 2006, pp 859–873
29. Gluck H (1975) Almost all simply connected closed surfaces are rigid. In: Geometric topology. Lecture notes in Math, vol 438. Springer, Berlin, pp 225–239
30. Roth B (1981) Rigid and flexible frameworks. Am Math Mon 88(1):6–21
31. Tay T, Whiteley W (1985) Generating isostatic frameworks. Topol Struct 11:21–69

32. Laman G (1970) On graphs and rigidity of plane skeletal structures. J Eng Math 4(4):331–340, October
33. Jacobs DJ, Hendrickson B (1997) An algorithm for two-dimensional rigidity percolation: the pebble game. J Comput Phys 137(2):346–365, June
34. Smith BS, Egerstedt M, Howard A (2008) Automatic deployment and formation control of decentralized multi-agent networks. In: Proceedings of the IEEE international conference on robotics and automation, Pasadena, 19–23 May 2008
35. Baillieul J, Suri A (2003) Information patterns and hedging brockett's theorem in controlling vehicle formations. In: Proceedings of the 42nd IEEE international conference on decision and control, Maui, December 2003, pp 556–563
36. Eren T, Whiteley W, Anderson BD, Morse AS, Belhumeur PN (2005) Information structures to secure control of rigid formations with leader–follower architecture. In: Proceedings of the 2005 American control conference, vol 4, Portland, June 2005, pp 2966–2971