Automatic Deployment and Formation Control of Decentralized Multi-Agent Networks

Brian S. Smith, Magnus Egerstedt, and Ayanna Howard

***Abstract*— Novel tools are needed to deploy multi-agent networks in applications that require a high degree of accuracy in the achievement and maintenance of geometric formations. This is the case when deploying distributed sensing devices across large spatial domains. Through so-called Embedded Graph Grammars (EGGs), this paper develops a method for automatically generating control programs that ensure that a multi-robot network is deployed according to the desired configuration. This paper presents a communication protocol needed for implementing and executing the control programs in an accurate and deadlock-free manner.**

I. Introduction

Formations, i.e. multi-robot configurations that satisfy certain geometric properties (e.g. [1], [2], [3]), can be represented through weighted graphs. In these graphs, the vertices represent the agents, while the weighted edges specify the corresponding inter-agent distances of the formation (e.g. [4], [5], [6]). In this paper, we use directed graph-encodings of the target formations. We assume that the graph contains the minimal number of edges to maintain the geometrical shape of the desired formation. For each edge in the formation, the responsibility of maintaining the distance is delegated to a single agent, denoted by the edge's direction. Thus, this directed graph represents a *minimally persistent formation* [7].

In this paper, we take these minimally persistent formations as inputs to an algorithm that automatically generates the appropriate control program for ensuring that the actual robots achieve and maintain the target formation. We achieve this by defining and executing *Embedded Graph Grammars* (EGGs) [8]. EGGs support the specifications of different control laws and the local network characteristics under which the control laws are applicable. We describe a method for defining an EGG that produces a persistent formation given a sequence of purely combinatorial graph operations that produce the formation. We also present a communication scheme called *prioritized lock negotiation* for implementing the resulting EGG on a distributed network of agents.

II. Preliminaries

Here, we review basic assumptions and terminology. We assume that the multi-agent team consists of n planar mobile robots, with $x_i(t) \in \Re^2$ being the position of agent i at time $t, i \in N = \{i, \ldots, n\}$. We moreover assume that the dynamics of each robot is given by a single integrator, i.e. $\dot{x}_i(t) = u_i(t), i \in N$. We also assume there is a defined *proximity range* $\Delta \in \Re$. A robot can identify and sense the relative position of other robots if and only if those robots are within proximity range. We moreover assume a user graphically inputs the desired *relative positions* $p_i \in \Re^2, i \in N$ of the planar robots to specify the target formation. As the robots are assumed to be homogeneous, it does not matter what robot is assigned to what position. We also assume that the formation is translationally and rotationally invariant.

The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: {brian,magnus,ayanna.howard}@ece.gatech.edu

A. Minimally Persistent Target Specifications

We let $\mathbb{G}$ be a minimally persistent graph that, along with the relative positions, defines the set of inter-agent distances to be kept and the geometry of the desired formation. In fact, $\mathbb{G}$ is a weighted, directed graph defined by the triple $(\mathbb{V}, \mathbb{E}, \delta)$, where $\mathbb{V} = \{\mathbb{V}_1, \ldots, \mathbb{V}_n\}$ is the vertex set, $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ is the edges set, and $\delta : \mathbb{E} \rightarrow \Re$ gives the edge weights as $\delta(\mathbb{V}_i, \mathbb{V}_j) = \|p_i - p_j\|$. As $\mathbb{G}$ is minimally persistent, it has been shown how to build up such a graph by a sequence of subgraphs [9]. This sequence starts with a graph with only two vertices, $\mathbb{G}_2$. We refer to this graph as the *leader-follower seed graph*. Here, one of the vertices is the leader and the other is the follower.

We assume that the next graph $\mathbb{G}_3$ in the sequence contains three vertices, obtained through a so-called *directed vertex addition*, where $\mathbb{V}_k$ is added, along with edges $(\mathbb{V}_k, \mathbb{V}_i)$ and $(\mathbb{V}_k, \mathbb{V}_j)$. Adding vertices in this systematic fashion results in a so-called *Henneberg sequence* [10] of nested subgraphs $\mathbb{G}_2, \mathbb{G}_3, \ldots, \mathbb{G}_n$, with $\mathbb{G}_n = \mathbb{G}$. We use the shorthand $\mathbb{V}(\mathbb{G}_p)$ and $\mathbb{E}(\mathbb{G}_p)$ to denote the vertex and edge sets of $\mathbb{G}_p$ respectively. The sequence itself can be automatically generated from $\mathbb{G}$ [9]. This paper assumes that the sequence $\mathbb{G}_2, \ldots, \mathbb{G}_n = \mathbb{G}$ is given, and produces automatically generated control laws that ensure that the actual robots achieve the target formation. The generation of these control laws is the main topic of the next section.

B. Robot Networks as Vertex-Labeled Graphs

An *Embedded Graph Grammar* (EGG) [8] is a formalism that encodes dynamic, geometric, and network properties of a multi-agent system in a unified manner. In this paper, we discuss how to construct an appropriate EGG for building up the desired target formation $\mathbb{G}$ through an assembly process based on the Henneberg sequence $\mathbb{G}_2, \ldots, \mathbb{G}_n$ from the previous sections. At the core of an EGG is the notion of a *graph-grammar* that takes as inputs vertex-labeled graphs and produces other vertex labeled graphs according to a given rule set. Through the application of the rules in the rule set,

edges may be removed or added to the graph, and the vertex labels may change.

For the development in this paper, the graph that we are interested in is one that keeps track of what actual robots have been assigned to what target positions in the formation, as well as what robots are in charge of maintaining what inter-robot distances. We denote this vertex-labeled graph by $G(t) = (V, E(t), l)$, where $V = \{V_1, \ldots, V_n\}$ is the vertex set, $E(t)$ is the edge set (at time t), and l assigns a label to each vertex. As a result, this graph is time dependent in that the vertex labels are dynamic and edges may be added as time progresses. Here, $G(t)$ corresponds to a graphical encoding of what the robots are actually doing at time t, while $\mathbb{G}$ encodes the target formation.

The vertex label function l assigns to each vertex in $G(t)$ either a vertex in $\mathbb{V}(\mathbb{G})$ or the unassigned label $w \notin \mathbb{V}(\mathbb{G})$. It also associates a Boolean value to each vertex $V_i \in V$ depending on whether or not the corresponding agent i has reached its target destination. We use the notation $l(V_i).assign \in \mathbb{V} \cup \{w\}$ to denote the desired position in the target formation that agent i has been assigned to, and $l(V_i).final \in \{true, false\}$ as a flag that indicates whether or not agent i has converged sufficiently close to its target destination. If $l(V_i).assign = w$, we say that agent i is a *wanderer*. Otherwise, we say that agent i is an *assigned* agent.

In the development of the EGG for assembling $\mathbb{G}$ sequentially, we need to define the initial condition for $G(t)$. We let $G(0) = (V, E(0), l_0)$, with $E(0) = \emptyset$, $l_0(V_i).assign = w$, and $l_0(V_i).final = false \ \forall i \in N$. This graph serves as the initial condition to a trajectory over $G(t)$ as the minimally persistent graph $\mathbb{G}$ is assembled, which is the topic of the next section.

III. Embedded Graph Grammars for Sequential Target Formation Assembly

A. Rules, Guards, and Control Laws

As the robots move around and establish links with neighboring robots, corresponding to distances that are to be maintained in order to produce the target formation $\mathbb{G}$, the network topology changes. In order to characterize this mechanism, we define graph-transition *rules*. Each rule consists of a vertex-labeled *left graph* L (the input to the rule), a vertex-labeled *right graph* R (the output to the rule), and a *guard* that defines the geometric conditions under which the rule is applicable. In order for rule r to be applicable to the robot network, some subset of $G(t)$ must "look" like $L \in r$. For this, we follow the notation in [8] and we define a *witness* $h : V_L \mapsto V$ as a label-preserving isomorphism between the vertices V_L of the left graph L and the vertices of $G(t)$. Witnesses formalize the notion of when two graphs "look" the same (including vertex labels and adjacencies). It is not enough that the left graph in the rule and a subgraph of $G(t)$ are isomorphic. We also require certain geometric conditions to be satisfied. These are encoded through a *guard function* $g : H \times (\Re^2 \times \cdots \times \Re^2) \mapsto \{true, false\}$, where H is the set of all witnesses for a specific rule. When a witness for a rule exists and the guard evaluates to $true$, we say that the guard is *satisfied* and the rule is *applicable*. If a rule is applicable, the subgraph of $G(t)$ isomorphic to V_L (denoted by $h(V_L)$) can be replaced in $G(t)$ by the right graph R in the rule. A guarded rule is represented by the triple $r = (L \rightharpoonup R, g)$. As a final building block, each assignment in the vertex labels (i.e. $l(i).assign$) corresponds to a particular control mode. In the remainder of this section, we define the specific rules and appropriate control modes that ensure that the target formation $\mathbb{G}$ is achieved.

B. Leader-Follower Rules

In Section II, we saw that the first subgraph in the Henneberg sequence was the leader-follower seed graph $\mathbb{G}_2$, with $\mathbb{V}(\mathbb{G}_2) = \{\mathbb{V}_{i_1}, \mathbb{V}_{i_2}\}$, and $\mathbb{E}(\mathbb{G}_2) = \{(\mathbb{V}_{i_2}, \mathbb{V}_{i_1})\}$. Due to the direction of the edge, $\mathbb{V}_{i_2}$ is in charge of ensuring that the proper distance is maintained between the vertices, and for that reason we call $\mathbb{V}_{i_1}$ the *leader* and $\mathbb{V}_{i_2}$ the *follower*.

Leader-Follower Position Rule

Through the leader-follower seed graph we can define a *leader-follower position rule* as $r^p_{lf} = (L^p_{lf} \rightharpoonup R^p_{lf}, g^p_{lf})$, where the left graph is given by the initial condition $L^p_{lf} = G(0) = (V, \emptyset, l_0)$ and the right graph is given by $R^p_{lf} = (V^p_{R_{lf}}, E^p_{R_{lf}}, l^p_{R_{lf}})$, with

$$
\begin{aligned}
&V^p_{R_{lf}} = V \\
&E^p_{R_{lf}} = \{(V_2, V_1)\} \\
&l^p_{R_{lf}}(V_i) = \begin{cases} (\mathbb{V}_{i_1}, true) & \text{if } V_i = V_1 \\ (\mathbb{V}_{i_2}, false) & \text{if } V_i = V_2 \\ (w, false) & \text{o.w.} \end{cases}
\end{aligned}
$$

Given a witness h for this rule, the guard g^p_{lf} evaluates to true if and only if corresponding robot $h(V_1)$ can detect and communicate with each robot in the network, i.e. $\|x_{h(V_1)} - x_{h(V_j)}\| < \Delta \ \forall V_j \in V$. Since the left graph is the initial graph $G(0)$, this implies that, initially, any robot within proximity range of all other robots can potentially be a leader, and any robot within proximity range to a potential leader is a potential follower.

As $l^p_{R_{lf}}(V_1).final = true$, i.e the leader agent has already achieved the desired position, the corresponding control law is simply $\dot{x}_{h(V_1)}(t) = 0$. We let the follower move according to

$$\dot{x}_{h(V_2)}(t) = x^\star_{h(V_2)} - x_{h(V_2)}(t),$$

where $x^\star_{h(V_2)}$ is the static target position given by

$$x^\star_{h(V_2)} = x_{h(V_1)} + \frac{\delta(\mathbb{V}_{i_1}, \mathbb{V}_{i_2})}{\|x_{h(V_2)}(t) - x_{h(V_1)}\|}(x_{h(V_2)}(t) - x_{h(V_1)}).$$

Leader-Follower Final Rule

As the follower is approaching the target position asymptotically, we also have a condition under which we consider the maneuver to be completed. For this we define the *leader-follower final rule*, $r^f_{lf} = (L^f_{lf} \rightharpoonup R^f_{lf}, g^f_{lf})$, whose only effect is that the label at vertex V_2 is changed from $false$ to $true$ when $\|x^\star_{h(V_2)} - x_{h(V_2)}(t)\| < \epsilon$, for a given threshold value $\epsilon > 0$.

C. Vertex Addition Rules

In Section II, we describe how vertex additions generate subsequent subgraphs $\mathbb{G}_3, \ldots, \mathbb{G}_n = \mathbb{G}$ of the Henneberg sequence. Assume that, for subgraph $\mathbb{G}_p$, there exists $\{\mathbb{V}_{j_1}, \mathbb{V}_{j_2}\} \in \mathbb{V}(\mathbb{G}_p)$ and a vertex addition operation adds vertex $\mathbb{V}_{j_3}$ to $\mathbb{V}(\mathbb{G}_p)$ and edges $\{(\mathbb{V}_{j_3}, V_{j_1}), (\mathbb{V}_{j_3}, V_{j_2})\}$ to $\mathbb{E}(\mathbb{G}_p)$ to produce subgraph $\mathbb{G}_{p+1}$. Due to the directions of these edges, $\mathbb{V}_{j_3}$ is in charge of ensuring the proper distance is maintained to vertices $\mathbb{V}_{j_1}$ and $\mathbb{V}_{j_2}$. This vertex addition operation defines a *vertex addition position rule* as $r^p_{va} = (L^p_{va} \rightharpoonup R^p_{va}, g^p_{va})$, where the left graph is given by $L^p_{va} = (V^p_{L_{va}}, E^p_{L_{va}}, l^p_{L_{va}})$, with

$$V^p_{L_{va}} = \{V_1, V_2, V_3\}$$
$$E^p_{L_{va}} = \begin{cases} \{(V_1, V_2)\} & \text{if } \exists (\mathbb{V}_{j_1}, \mathbb{V}_{j_2}) \in \mathbb{E}(\mathbb{G}) \\ \{(V_2, V_1)\} & \text{if } \exists (\mathbb{V}_{j_2}, \mathbb{V}_{j_1}) \in \mathbb{E}(\mathbb{G}) \\ \emptyset & \text{o.w.} \end{cases}$$
$$l^p_{L_{va}}(V_j) = \begin{cases} (\mathbb{V}_{j_1}, true) & \text{if } V_j = V_1 \\ (\mathbb{V}_{j_2}, true) & \text{if } V_j = V_2 \\ (w, false) & \text{if } V_j = V_3 \end{cases}$$

and the right graph is given by $R^p_{lf} = (V^p_{R_{va}}, E^p_{R_{va}}, l^p_{R_{va}})$, with

$$V^p_{R_{va}} = \{V_1, V_2, V_3\}$$
$$E^p_{R_{va}} = \{(V_3, V_1), (V_3, V_2)\} \cup E^p_{L_{va}}$$
$$l^p_{R_{va}}(V_j) = \begin{cases} (\mathbb{V}_{j_1}, true) & \text{if } V_j = V_1 \\ (\mathbb{V}_{j_2}, true) & \text{if } V_j = V_2 \\ (\mathbb{V}_{j_3}, false) & \text{if } V_j = V_3 \end{cases}$$

Given a witness h for this rule, the guard g^p_{va} evaluates to true if an only if the corresponding robot $h(V_3)$ is close enough to $h(V_1)$ and $h(V_2)$ to be able to detect them, i.e. $\|x_{h(V_3)} - x_{h(V_1)}\| < \Delta$ and $\|x_{h(V_3)} - x_{h(V_2)}\| < \Delta$.

Since this rule assigns edges only to $h(V_3)$, we must define a control law for $\dot{x}_{h(V_3)}(t)$ based on $x_{h(V_1)}(t)$ and $x_{h(V_2)}(t)$. Note that these three vertices $h(V_1)$, $h(V_2)$, and $h(V_3)$ have been assigned to vertices $\mathbb{V}_{j_1}$, $\mathbb{V}_{j_2}$, and $\mathbb{V}_{j_3}$ in $\mathbb{V}(\mathbb{G})$, respectively. Therefore, positions $\{p_{j_1}, p_{j_2}, p_{j_3}\}$ from Section II define the relative geometry that is desired for the corresponding robots. Robot $h(V_3)$ can determine its target position by $x^\star_{h(V_3)}(t) = f(x_{h(V_1)}(t), x_{h(V_2)}(t), p_{j_1}, p_{j_2}, p_{j_3})$ where f performs the corresponding translation and rotation. We let this robot move according to

$$\dot{x}_{h(V_3)}(t) = x^\star_{h(V_3)}(t) - x_{h(V_3)}(t).$$

If we assume that $\mathbb{V}_{j_1}$ and $\mathbb{V}_{j_2}$ are the leader and follower, then the leader-follower rule implies that they converge to the appropriate distance, which implies that agent $h(V_3)$ converges to the desired geometric relationship to $h(V_1)$ and $h(V_2)$. By induction, this implies that all robots added by a vertex addition rule converge to their appropriate geometric relationship with robots $h(V_1)$ and $h(V_2)$ in their respective witnesses.

Vertex Addition Final Rule

As robot $h(V_3)$ is approaching the target position asymptotically, we also have a condition under which we consider the maneuver to be completed. For this we define the *vertex addition final rule* $r^f_{va} = (L^f_{va} \rightharpoonup R^f_{va}, g^f_{va})$, whose only effect is that the label at vertex V_3 is changed from $false$ to $true$ when $\|x^\star_{h(V_3)}(t) - x_{h(V_3)}(t)\| < \epsilon$, for a given threshold value $\epsilon > 0$.

D. Wander Mode

The previous rules allow robots to achieve desired geometric relationships. However, the guards for these rules depend on robots labeled w satisfying geometric conditions with robots in the rule witnesses. Therefore, we require a mode that ensures that robots labeled w eventually satisfy these constraints. We call this mode *wander mode*.

To implement wander mode, each assigned robot with a label $\mathbb{V}_i \in \mathbb{V}(\mathbb{G})$ is given a *hop-counter* λ, which it communicates to all assigned robots within proximity range. We set to zero the hop-counters of all assigned robots whose labels allow them to participate in vertex addition position rules that have not been applied. This implies that their labels occur in the left graph of position rules that have not yet been applied (This also requires robots to "keep track" of what rules have and have not been applied, an issue that is discussed in the following section). Assume that robot i is an assigned robot that cannot participate in a vertex addition position rule. Assume that Λ_i is the set of all the hop-counters of all robots within proximity range of i. Then we define robot i's hop-counter by

$$\lambda_i = \begin{cases} \min(\Lambda_i) + 1 & \text{if } \min(\Lambda_i) < n \\ n & \text{o.w.} \end{cases} \quad (1)$$

Since all robots with hop-counters equal to zero have been assigned positions in $\mathbb{G}$, this implies that there always exists a "path" of assigned robots with decreasing hop-counters that leads to a hop-counter of zero, if such a robot exists. Therefore, wander mode is defined so that wanderers perform circular motion around the robot with the lowest hop-counter in proximity range. When a robot with a lower hop-counter comes into its perception, it switches to perform circular motion around this robot. This process repeats until the wanderer finds an assigned robot with a hop-counter equal to zero. The circular motion is performed with a radius equal to the largest weight of the edges in $\mathbb{E}(\mathbb{G})$. This guarantees that the wanderer finds the next robot in the path, since the weight of all edges in $\mathbb{E}(\mathbb{G}) < \Delta$ and the next robot in the path must be within Δ of the current robot the wander is circling.

Once a wanderer encounters an robot whose hop counter equals zero, it enters an exclusive partnering relationship with that robot. The assigned partner i changes its hop-counter from zero to $\min(\Lambda_i) + 1$. The assigned partner also refuses any more partnerships with other wanderers. Since all edges $\mathbb{E}(\mathbb{G}) < \Delta$, as a wanderer circles its partner at a radius equal to the largest weight in $\mathbb{E}(\mathbb{G})$ it satisfies the guards of any potentially applicable rule. As each rule is applied, the robots involved in the rule application reevaluate their hop-counters as defined in (1).

Note that each vertex addition rule has two vertices with labels that assign a hop-counter of zero to assigned robots.

This implies that two wanderers can potentially be partnered with different robots, but for the same rule. Therefore, if the hop-counter changes from zero to another value, this signals any partnered wanderers to abandon the partnership and to follow a path to another robot with a zero hop-counter. Since this situation only occurs when all robots required for a vertex addition rule are present, then this implies that the redundant partnered wanderer is always freed, and can proceed towards another vertex addition rule opportunity. The definition of hop-counters also implies that, when all robots that can participate in vertex additions have partners, there may be intervals of time where there is no hop-counter equal to zero. However, this situation guarantees that a vertex addition rule is applied, since all assigned robots that can participate in vertex additions have a partnered wanderer. Eventually, vertex addition rules assign positions to wanderers.

IV. Rule Evaluation and Communication

This section discusses the implementation of this EGG on a network of robots, in terms of rule evaluation and communication. We assume that the label and adjacency information is distributed across the network such that each robot has immediate access only to its own label and adjacency information. The label and adjacency information corresponding to other robots can only be obtained through communication. The robots in the network must change modes and execute control laws in a manner defined by the EGG's guarded rules, labels, and the corresponding control law for each label. For the EGG we have defined, this also requires the network to guarantee that no rule is applied more than once to prevent redundant position assignments, and that robots keep track of what rules have not been applied to effectively update their hop-counters. Since this is a decentralized network, this implies that robots must negotiate rule applications in a manner consistent with the EGG.

A. Primaries and Rule Evaluation

For each rule $r = (L \rightharpoonup R, g)$, there is an assigned vertex $V_i \in V_L$ of the left graph L such that the guard function requires that $\|x_{h(V_i)}(t) - x_{h(V_j)}(t)\| < \Delta \ \forall V_j \in V_L$. We define V_i as the *primary vertex* of the rule. For leader-follower position rules, this is the vertex corresponding to the leader robot. For vertex addition position rules, this is the vertex corresponding to the wanderer in the left graph. For final rules, this is the vertex with the $false$ final label.

When a witness exists that maps a rule's primary vertex to a robot's vertex in $G(t)$, we say that the robot is a *primary robot*. Since the primary robots are within proximity range of each robot corresponding to the witness of an applicable rule, then these robots can obtain all the local graph information necessary to apply a rule to a subgraph of the embedded graph and, thereby, modify that information in a manner defined by the applicable rule. Because all the primary vertices of each rule correspond to robots in wander mode, we insist that only wanderers attempt to apply rules, and only with witnesses that map them to the primaries of the rules. Each wander robot determines whether or not it is a primary by requesting the local graph information of its neighbors and comparing it to the left graphs of the rules to see if one is applicable. If so, then the primary robots attempt to apply the rules to the embedded graph by modifying the local graph information of its neighbors. This process is called *rule evaluation*.

B. Prioritized Lock Negotiation

It is necessary that each primary robot has exclusive control of all robots necessary to apply a rule; if not, then it is possible for multiple primary robots to modify the graph information in a manner inconsistent with the rules, or apply a rule more than once, producing *graph inconsistencies*. Graph inconsistencies occur when there exists subgraphs of $G(t)$ that are not intended to exist by the EGG design. To prevent graph inconsistencies, we define a *prioritized lock negotiation* communication scheme. This scheme gives primaries exclusive control of other robots' EGG information through a series of *lock negotiations*. When a primary robot wants to apply a rule involving another robot, it performs a *lock request* for that robot. If the robot being requested for a lock is *unlocked*, it accepts the lock of the primary and records the primary's index. The locked robot refuses any lock requests while locked. Once locked, the locked robot allows the owner of its lock to modify the locked robot's EGG information.

Once a primary has locked the entire set of robots necessary for the rule application, it verifies that the rule is still applicable, i.e. the graph information is still consistent with the rule, and the rule has not been applied. Since each robot has its own copy of the rule set, we exploit the locality of the guarded rules to prevent any rule from being applied more than once. As each rule is applied, it is removed from the rule sets of the robots involved in the application. Then, before a primary can apply a rule with its locked robots, it must first verify that each locked robot has the rule in its rule set. Since each robot removes the leader-follower positions rules from their set of rules as it is applied (with the entire graph), this guarantees that the leader-follower position rule is applied only once. Similarly, when vertex addition rules are applied, the corresponding vertex addition position rule is removed from the rule sets of the involved robots. The vertex addition rule cannot be repeated, since the involved robots have already removed it from their rule sets. This implies that all rules are never applied more than once. This also allows the assigned robots to have accurate knowledge of which rules can still be applied, which they use to determine the hop-counters. When the primary has completed all modifications of graph information necessary to apply the rule, it then *unlocks* all the robots it has locked.

With many primary robots attempting to lock sets of other robots, it is possible for primary robots to lock robots in a manner that prevents any applicable rule from being applied. We define this as *deadlock*. To prevent deadlock, we define a priority to each robot that corresponds to its index. We

say that robot i has a higher priority than robot j if $i < j$. Since each robot has a unique index, no robots have the same priority. When a locked robot refuses a lock request, it communicates the index of the primary that locked it to the robot requesting the lock. If it has a higher priority than the robot that owns the lock, it immediately retries the lock request. If it has a lower priority than the robot that owns the lock, it immediately unlocks all robots that it owns locks for, and waits for a time τ before reattempting the rule.

We assume that τ is defined as a worst-case period of time long enough to allow n robots to attempt n rule negotiations in series. When robots compete for locks, there is always a lowest priority robot. If no robot can acquire a lock to all the robots involved in a rule application, then the lowest priority robot always releases its locks and waits for time τ before trying again. Even if more robots begin attempting to compete for the same locks, the delay time τ is defined for a worst-case scenario of n robots competing. This implies that, in a worst-case scenario, there will eventually be only one robot attempting to acquire locks. Therefore, the network cannot be constantly deadlocked.

V. Implementation Scenario

To consider the application of this EGG on a network of mobile robots, we assume the following scenario: We have a network of n robots with data collection sensors, and we wish to distribute them in a 5 m triangular coverage pattern over an area of interest. We assume that the robots have a proximity range of $\Delta = 6$ m. We enter a triangulation pattern of positions in our graphical program discussed in section II and shown in Fig. 1.

This graphical program allows us to enter the proximity range and, using the algorithms presented in [9], determines that the formation is persistently feasible and defines the minimally persistent graph $\mathbb{G}$ shown in Fig. 1, as well as a leader-follower seed $\mathbb{G}_2$ (here, with vertices 1 and 2), and a sequence of vertex addition operations that define a Henneberg sequence $\mathbb{G}_2, \ldots, \mathbb{G}_n = \mathbb{G}$. Using the methods previously described, the program generates the EGG defined by $\mathbb{G}$. We assume the robots are programmed to implement this EGG, along with prioritized lock negotiation, and are positioned in the area of interest such that at least one robot is within proximity range of all robots. Then the EGG is executed.

VI. Results

The scenario discussed in Section V is simulated for n =7 and n =25 robots, as well as numerous random target formations, including random numbers of robots and random edge weights. In each simulation, the resulting states of the robots are checked to make sure that all robots are within proximity range of any robots necessary for control calculations. Also, the rule evaluation simulates the prioritized lock negotiation. During simulation, witnesses are exhaustively searched, and when rules are applicable, each primary robot in the network is assigned a corresponding witness to attempt to apply, if one exists for that robot. In this way, we attempt to maximize

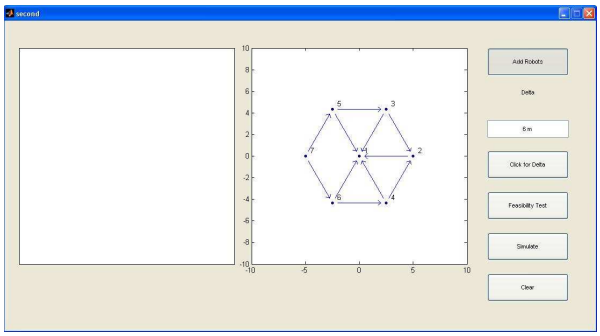

Fig. 1. A graphical program derives $\mathbb{G}$ from a set of desired positions entered by a user, representing a desired formation. This program also simulates the network in the left plotting area.

the number of robots competing for locks. The lock requests are assigned a random order to arrive at their corresponding robots.

Fig. 2 shows the simulation results of the scenario in Section V where n =7. The graphical program in Fig. 1 simulates the network in the left window, from which Fig. 2 and Fig. 3 are taken. In Fig. 2(a), we see that at least one wanderer is in proximity range of all robots in the network. This implies that the leader-follower position rule is applicable, and it is applied, shown in Fig. 2(b). In these figures, the numbers correspond to the vertex indices in Fig. 1, and a dash ("-") indicates that the $final$ field is $false$. While vertex addition operations define a sequence of subgraphs, many vertex addition rules can be applied concurrently. As shown in Fig. 2(c), two vertex addition position rules are applied simultaneously, before either has been finalized. This is because these rules depend only on the presence of two assigned vertices in $G(t)$, not on the entire subgraph before the corresponding vertex addition operation. In this way, EGGs can take advantage of concurrency to accomplish the formation task. The wanderers spend most of their time in wander mode circling the leader robot 1 because this robot's label is in the left graph of every vertex addition position rule. Finally, the EGG has successfully completed, as shown in Fig. 2(h).

To demonstrate further the impact of concurrency and the effectiveness of the wander mode, we implemented a similar scenario with n =25 robots, shown in Fig. 3. Using the same proximity range Δ and edge weights in the previous scenario. As rules are executed, and wanderers begin satisfying the guards of vertex addition rules, more and more rules are able to apply concurrently. Eventually, the large triangulation pattern is completed.

To verify these results, many thousands of randomly generated positions were used to define minimally persistent graphs, vertex addition operations, and EGGs. Experiments show that the most "risky" formations are those where edge weights are close to the proximity range Δ. Since the final rules in Section III switch the $final$ label to $true$ when the robots are within ϵ of there desired positions, then ϵ defines a maximum error that can be present before vertex additions

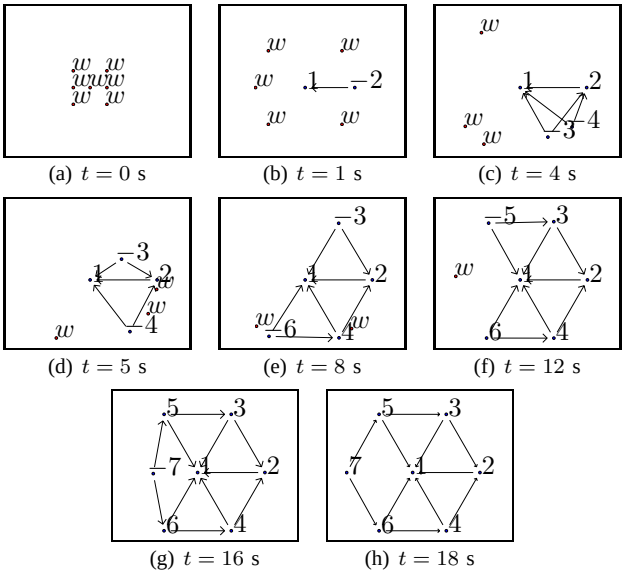

Fig. 2. EGG execution to assemble a hexagon triangular coverage pattern.

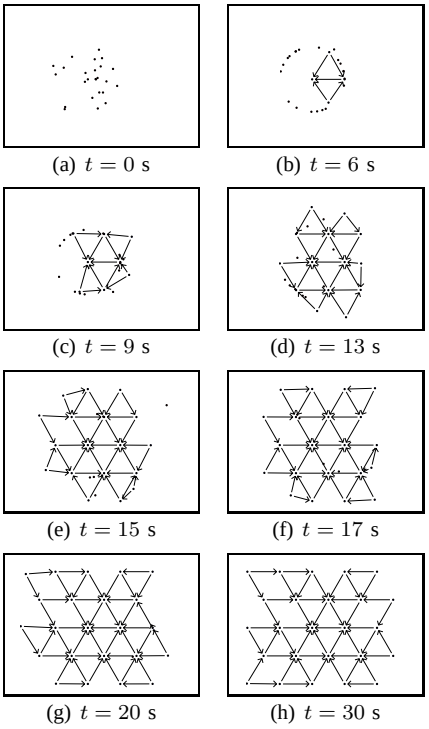

Fig. 3. EGG execution to assemble a large triangular coverage pattern.

are performed. If ϵ is too large, and edges have weights close to Δ, it is possible that enough error is present to force a robot to be outside of proximity range of the robots necessary for its control calculations. This situation was monitored in simulation. Experiments show that when this occurs, it is possible to redefine ϵ to be smaller in a manner such that this does not occur in repeated execution. Experiments also show that it is always possible to define τ such that the prioritized lock negotiation never deadlocks.

In practice, the parameters ϵ and τ are determined by the robot hardware involved. Typically, ϵ is minimally defined to adequately represent when the robot has driven sufficiently close to its goal to indicate to other robots that its maneuver is completed. This varies with the precision of the sensor hardware, and making it as small as possible helps guarantees that the areas within proximity range of assigned robots do not change quickly. Also, τ is both a function of the network size and the robot's communication hardware. Typically, τ is estimated by determining the time required for robot negotiations and how it scales with network size. However, it is only necessary to define it sufficiently large.

VII. Conclusions

Given minimally persistent target formations and Henneberg sequences corresponding to vertex additions, we present automatic tools to generate EGGs that allow these formations to emerge in a network of mobile robots. This includes a description of graph-based representations of target formations and the multi-agent network of robots, as well as rules for specifying changes in network topology and the control modes of the individual robots. We also present a communication scheme that enables a distributed network of robots to implement these EGGs effectively, in a manner that both guarantees the accuracy of the EGG implementation as well as the avoidance of deadlock. These methods are verified in simulation for a variety of target formations and network sizes. All experiments demonstrate that it is possible to define control laws and communication scheme parameters to achieve these goals for minimally persistent target formations generated by Henneberg sequences of vertex additions.

References

[1] P. Ogren, M. Egerstedt, and X. Hu, "A control lyapunov function approach to multi-agent coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 847–851, Oct 2002.

[2] G. A. Kaminka and R. Glick, "Towards robust multi-robot formations," in *Conference on International Robotics and Automation*, 2006, pp. 582–8.

[3] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 637–49, August 2006.

[4] T. Eren, W. Whiteley, B. D. O. Anderson, A. S. Morse, and P. N. Belhumeur, "Information structures to secure control of rigid formations with leader-follower architecture," in *Proceedings of the American Control Conference*, Portland, Oregon, June 2005, pp. 2966–2971.

[5] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, June 2003.

[6] M. Ji and M. Egerstedt, "Distributed coordination control of multi-agent systems while preserving connectedness," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 693–703, Aug 2007.

[7] J. M. Hendrickx, B. D. O. Anderson, J.-C. Delvenne, and V. D. Blondel, "Directed graphs for the analysis of rigidity and persistence in autonomous agent systems," *International Journal of Robust and Nonlinear Control*, 2000.

[8] J. M. McNew and E. Klavins, "Locally interacting hybrid systems with embedded graph grammars," in *Conference on Decision and Control*, 2006, to Appear.

[9] B. Smith, M. Egerstedt, and A. Howard, "Automatic generation of persistent formations for multi-agent networks under range constraints," in *Proceedings of the International Conference on Robot Communication and Coordination*, 2007.

[10] L. Henneberg, "Die graphische statik der starren systeme," 1911.