A Graph Grammar Approach to Self-Organizing Robots in Geometric Formations

Sérgio Manuel da Silva

Abstract—A graph grammar approach that may be used for the distributed assembling of geometric formations of robots is presented in this paper. The focus is on the synthesis of binary graph grammars, that is, sets of rules involving at most two robots, which give the assembly steps for unorganized groups of robots converge to formations with geometric constraints. Making each robot a local interpreter of such rules a formation is able to grow dynamically in a decentralized manner from simple pairs of constituents to larger and more complex arrangements. A suitable algorithm was implemented to generate binary graph grammars with constructive and destructive rules for any geometric formation defined by a directed spanning tree. An example of the use a simple binary graph grammar to rule three mobile robots self-organizing in a line formation is shown in the end.

Keywords—Binary graph grammar, binary rule, directed spanning tree, geometric formation, leader-follower, robot.

I. Introduction

The motivation of this and other works involving formations of robotic systems is grounded on the fact that the union makes the strength. And the nature is rich in such fact, where the strength may have several meanings. For instance, herd and pack animals use group organization to attack larger prey, defend against predators and increase their chances of survival. Dolphins are known to swim in formations to protect their calves from predators. Also, geese and cranes fly in wedge formation to reduce the individual drag force (due to the formation of wing tip vortices) and to communicate more easily with good visual contact of each other to keep the flock together. Inspired in such biological facts, researchers have been developing several works on the control of fleets of mobile robots (terrestrial, underwater and aerial vehicles) aiming to find applicable solutions in real situations to improve people’s quality of life.

Among works where tools from graph theory and control theory have been merged to control formations of autonomous vehicles those presented in [1] and [2] are good examples. The theory of directed graphs is used to deal with inter-vehicle communication constraints, where vertices represent vehicles and edges represent communication links. And, the relationship between the location of the Laplacian eigenvalues and the graph structure is strictly related to the stability of a given geometric formation. Also, in [3] and [4], graph concepts are used to self-organize robotic systems in a distributed manner. Although vertices of a graph represent parts or vehicles, whose states correspond to discrete symbols (labels), the presence of an edge between two vertices represents the fact they are attached (with a topological relationship). The problem of controlling a formation of robot systems is solved in two steps: i) the synthesis of a set of rules, called a graph grammar, and ii) the interpretation of that grammar. The authors present two algorithms to synthesize sets of rules. One takes as an argument an unlabeled tree and returns a set of binary rules. The other takes as an argument an arbitrary unlabeled graph and returns ternary rules, with better results at the convergence level. They also present two simulation examples where graph grammars can be used: in the first one artificial potential field functions are used to attract or repel self-motive robots, if they belong to a set where at least one rule match or to a set where no rules match, respectively; in the second one a large number of robotic parts float in a stirred fluid, and a collision by chance of two parts causes an attachment if a rule is applicable.

Similarly to the latest two referred works, a way to make robots self-organize in formations without centralized coordination and using only local sensing (each robot is only required to sense its nearest neighbor) is presented here. However, the approach is much closer to another interesting work in the area, [5], because the considered formations have geometric constraints. The formation graphs are restricted to directed spanning trees, and only binary rules are considered (rules involving at most two robots). Based on these rules a formation is able to grow in a bottom-up manner using a kind of negotiation via a simple broadcast protocol. Singletons, individual robots that are not part of an existing formation, negotiate with other singletons to form leader-follower pairs (line formations of two elements that have changed their role). As leader-follower pairs encounter other leader-follower pairs, or singletons, they negotiate and become larger assemblies. Although narrowed attention is given to the synthesis of binary graph grammars, a simple and intuitive example of interpretation of a binary graph grammar to assemble a line formation of three mobile robots is also presented.

The rest of the paper is organized as follows. In Section II the basis of graph grammars with application in the self-organization of robotic systems is addressed. Section III provides an overview of the approach followed here, that is, a way to synthesize binary graph grammars to self-organizing robots in geometric formations. Section IV provides results of simulations where three mobile robots are ruled by a binary graph grammar to self-assemble in a line formation. Finally, section V is reserved to conclusions and to comments on future work.

Sérgio M. da Silva is with Instituto de Sistemas e Robótica (ISR), Lisbon, Portugal, and with the Electrical Engineering Department, Instituto Politécnico de Leiria - ESTG, Leiria, Portugal. E-mail: ssilva@estg.ipleiria.pt.
II. Graph Grammars

A. Definitions

Graph grammars are a generalization of the standard grammars used in linguistics and in automata theory. This is not a new topic, but its use to model distributed assembly started recently with the pioneer work [3]. It was decided to adopt here the same definitions to formally describe the current state of a formation assembly.

A simple labeled graph over an alphabet $\Sigma$ is a triple $G = (V, E, l)$ where $V$ is the set of vertices, $E$ is the set of edges, and $l : V \rightarrow \Sigma$ is a labeling function. Usually the alphabet $\Sigma = \{a, b, c, \ldots\}$ is used.

Given graphs $G_1$ and $G_2$, $f : G_1 \rightarrow G_2$ and $f : V_{G_1} \rightarrow V_{G_2}$ are written equivalently to mean that $f$ is a function from the vertex set of $G_1$ to the vertex set of $G_2$. A function $h : G_1 \rightarrow G_2$ is a label preserving embedding if: 1) $h$ is injective, 2) $\{x, y\} \in E_{G_1} \Rightarrow \{h(x), h(y)\} \in E_{G_2}$, 3) $l_{G_1} = l_{G_2} \circ h$. If $h$ is also surjective then it is called an isomorphism. The graphs $G_1$ and $G_2$ are said to be isomorphic (written $G_1 \simeq G_2$) if there exists an isomorphism relating them.

A rule is a pair of graphs $r = (L, R)$ where $V_L = V_R$. The graphs $L$ and $R$ are called the left hand side and right hand side of $r$ respectively. The size of $r$ is $|V_L| + |V_R|$. Rules whose vertex sets have one, two and three vertices are called unary, binary and ternary, respectively. Rules may be constructive ($E_L \subset E_R$), destructive ($E_L \supset E_R$) or mixed (neither constructive or destructive). A rule is acyclic if its right hand side contain no cycles (the left hand side may contain cycles). A rule $r$ is applicable to a graph $G$ if there exists an embedding $h : L \rightarrow G$. In this case the function $h$ is called a witness. An action on a graph $G$ is a pair $(r, h)$ such that $G$ is applicable to $G$ with witness $h$.

Given a graph $G = (V, E, l)$ and an action $(r, h)$ on $G$ with $r = (L, R)$, the application of $(r, h)$ to $G$ yields a new graph $G' = (V', E', l')$ defined by:

$$
V' = V
$$
$$
E' = (E - \{(x, y) \mid \{x, y\} \in L\}) \cup \{(h(x), h(y)) \mid \{x, y\} \in R\}
$$
$$
l'(x) = \begin{cases} l(x) & \text{if } x \notin h(V) \\ l(r) \circ h^{-1}(x) & \text{otherwise} \end{cases}
$$

An initial state graph $G_0$ along with a set of rules defines a system known as an assembly. Rule sets provide a concise grammar (graph grammar) to describe the transitions from one partial assembly state to another ($G_0 \rightarrow G_1 \rightarrow G_2 \ldots$). Since many rules can have the same witnesses and can be applied in any order or even in parallel to each other, assembly systems are nondeterministic finite automata.

B. A Synthesis Algorithm

A simple recursive algorithm for generating a constructive set of rules initially given by [3] is shown in figure 1. The algorithm takes as an argument an unlabeled tree $(V, E)$ and returns a pair $(\Phi, l)$ where $\Phi$ is a set of constructive binary rules and $l$ is a labeling function on $V$.

```
1. Require $T = (V, E)$ is an unlabeled tree
2. if $V = \{x\}$ then
3. return ($\emptyset, \lambda z. \text{if } z = x \text{ then } a \text{ else } \perp$ end if)
4. else
5. let $(x, y)$ be an edge of $(V, E)$
6. let $(V_1, E_1)$ be the component of $(V, E - \{x, y\})$ containing $x$
7. let $(V_2, E_2)$ be the component of $(V, E - \{x, y\})$ containing $y$
8. let $(\Phi_i, l_i) = \text{MakeTree}(V_i, E_i)$ for $i = 1, 2$
9. let $\Phi = \Phi_1 \cup \Phi_2 \cup \{(l_1(x))_2(y) \Rightarrow u - v\}$
10. let $l = \lambda x. \text{if } x = z \text{ then } u \text{ else } v$
11. else if $x = y \text{ then } v$
12. else if $l_1(x) \neq \perp \text{ then } l_1(x)$
13. else $l_2(z)$
14. return $(\Phi, l)$
15. end if
16. end if
```

Notice that the tree is considered unlabeled, but it is necessary to have an initial identification of the vertices, for instance by numerating them, which does not conflict with the used alphabet $\Sigma$. It starts by decomposing the formation tree (steps 1-2) until all edges have been removed. At this stage all vertices are given an initial labeling ($\lambda a.$ is the lambda calculus notation for the function assigning the label $a$ to all vertices). As the algorithm back steps out of the recursion started in step 4, it replaces each edge $(x, y)$ and generates a single constructive rule for it (line 9). The algorithm also updates the label mapping function with two new labels $u$ and $v$ that have not been used before by any recursive call (lines 10-14). When the algorithm finishes, the cumulative result (line 15) is a set of rules (exactly $|V| - 1$ rules) as well as a final label set (exactly $2|V| - 2$ labels) for the completed formation. Since every label associated with each vertex is unique, the resulting rule set will be stable.

C. Interpretation of a Graph Grammar

Besides the synthesis of graph grammars, it is also necessary to interpret rules at the workspace level, which is obviously a distinct thing. First, it is necessary to embed a graph to the workspace, i.e., to associate each workspace position to each vertex in the graph that denotes the corresponding robot. Second, a continuous guard has to be associated to each grammatical rule that states what condition on, for example, the locations of robots must hold for the rule to be applicable. Third, motion controllers have to be associated to each robot.

D. Example of a Binary Graph Grammar

Although rules can involve any number of vertices, the focus will be on a class of rules that involves exactly two vertices. These binary rules will create or destroy no more than a single edge and/or modify the two labels of the vertices involved. A simple short hand notation for a constructive binary rule is given by $a a \rightarrow b - c$, where the left hand side represents the witness and the right hand side the action. In this case two vertices are said to conform to each other if one is labeled $a$ and the other labeled $a,$
and neither maintains a constraint with the other. If the rule is applied then the new graph will have these vertices labeled \( b \) and \( e \) respectfully and a new edge will be created between them. Similarly, a short hand notation for a destructive binary rule is given by \( b \rightarrow c \rightarrow a a \), and for a mixed rule is given by \( d \rightarrow e \rightarrow f \rightarrow g \).

The rule set (2) is a possible example of a binary graph grammar where the three types of rules are found.

\[
\Phi = \begin{cases} 
  a a \rightarrow b - c \\
  a c \rightarrow d - e \\
  a f \rightarrow h - i \\
  b - c \rightarrow a a \\
  d - e \rightarrow f - g 
\end{cases} \tag{2}
\]

For instance, using this grammar to assemble a formation of four robotic systems may result like presented in figure 2. Initially all robots are in state \( a \) and are at positions resembling the vertices of a square. It is also assumed that each robot can only communicate with their adjacent partners and possesses an electromagnetic system to make physical attachments. Clearly, the only rule that can be used initially is the first one (constructive), and twice. Two pairs of robots are thus attached (could be vertical attachments instead of horizontal ones). After that, a communication failure between one pair of robots is assumed to occur and they unattach, changing their states to match the fourth rule (destructive). Now, the second rule (constructive) is applicable, another attachment is done and the formation reaches a state where none constructive rule can be applied. Therefore, one pair of attached robots changes its state remaining attached by the unique mixed rule of the set (the last one). The final formation is achieved by the third rule (constructive).

![Example of a formation assembly ruled by the binary graph grammar](image)

Although the final formation in figure 2 resembles a square with an open edge as it was desired or expected, that open edge is considered to be achieved occasionally. For instance, if vertical attachments had happened in the first constructive step, or if the communication failure never had occurred, the final result would a different one.

III. The Approach

A. Overview

Since the objective of a graph grammar is to be interpreted by parts or vehicles in an assembling process, it is clear that the vertices of the formation graph correspond to parts or vehicles, and edges to topological or geometric relationships between them. As in [5] a directed spanning tree is used here to represent a given geometric formation. It is a subgraph of a directed graph containing all the vertices and where any two vertices are connected by exactly one directed path. Directed means that edges are ordered pairs of vertices \((x, y)\), where \( y \) is called the head and \( x \) the tail. In this manner a sufficient number of desired geometric constraints is explicitly represented, where each edge is also appropriately labeled to identify the distance between two vertices as well as the bearing as seen from the tail to the head. Therefore is now clear that the labels associated to the vertices may represent different classes of states of the robots in assembling. These classes are broadly classified here into four categories:

**Leader** - This class of states represents robots that do not have any constraints to maintain. Any partial assembled formation will have a single robot in this state. While in this state a robot will wander in search of others. It is the overall responsibility of the robots in this state to gather enough followers to correctly assemble the formation.

**Follower** - This class of states represents robots that are part of a formation yet are not a leader. Their primary function is to maintain a fixed bearing and distance to a single target.

**Singleton** - This state is a special case of the leader class. All robots are initialized to this state. Like other leader states, singletons do not maintain any constraints and wander in search of other singletons or partial assembled formations.

**Formation Leader** - Another special case of the leader class is the formation leader. Depending on the rule set, this state is often unique and only obtained by a single robot when a formation has been completed.

The approach to formation control is to dynamically grow a formation from simpler constituents. Each robot starts as a singleton, which does not follow other robots, nor it is followed. As singletons encounter other singletons, they negotiate via a simple broadcast protocol. The outcome of these negotiations is typically a role change for both robots involved. Two singletons leave the negotiating phase as a leader-follower pair. The leader-follower pair represents a line segment of length two. As leader-follower pairs encounter other leader-follower pairs (or singletons) they negotiate and fuse into larger assemblies. The current state of the formation assembly is maintained across all robots participating in the formation. Each robot maintains a single state variable that represents its role and its relative position in the formation. This state variable decides how the robot will conform, or how it will react to future negotiations with other robots. The rules of how the various sub assemblies conform to one another are maintained as a state transition table in each robot and are described by a grammar.
B. Synthesis Algorithm

Slight changes were done to make the algorithm of figure 1 work well with directed spanning trees. The line before line 1 was replaced by "Require $T = (V, E)$ is an unlabeled directed spanning tree". Line 8 was replaced by "let $u,v$ be new unique labels from the leader and from the follower classes, respectively". Also, before giving all vertices an initial labeling, all the edges are also labeled to represent the constraints between leader-follower pairs (the alphabet $\Sigma_E = \{A,B,C,\ldots\}$ is used). Finally, line 15 is changed to make the algorithm return a cumulative set of rules with the appropriate edge labels attached to the right hand side of the rules (attached to the labels which represent states of followers). It is strictly necessary to have this new labeling in the rules at the interpretation level. In fact, as a robot becomes a follower it has to know the geometric constraint with the leader its controller has to guarantee.

The algorithm was implemented in Matlab environment. It takes two vectors as arguments, one with the edges (order pairs of numbered vertices), and the other with the corresponding cartesian coordinates. Slight changes may be done to input the edges in matricial form (using adjacency or incidence matrices). The output is a graphical evolution of the synthesis process, as well as the rule set in text mode. A snapshot of the graphical evolution of the synthesis process for a wedge formation of five robots is shown if figure 3. In figure 3(a) the edge-removing takes place, where the first two target edges are those which involve the vertex corresponding to the formation leader. In figure 3(b) the edge-restitution is done sequentially in the inverse order, and a rule is added to the rule set at each step. As it was expected, the final rule set (3) has four rules with ten distinct labels in the right hand sides, including already the labels A and B to represent the two distinct constraints found.

$$\Phi = \begin{cases} a a \rightarrow b - c A \\ a a \rightarrow d - e B \\ a b \rightarrow f - g A \\ f d \rightarrow h - i B \end{cases}$$

(3)

Notice that the rule set (3) corresponds only to a single possible path, from the initial graph to the final one. The implemented algorithm also computes and outputs the remaining eleven rule sets. Notice that in this case there are twenty four possible permutations for edge-restitution ($|E|!$), but half of them are repeated, i.e., they represent isomorphic final graphs.

C. Improved Synthesis Algorithm

Although the rules set generated by the synthesis algorithm of the previous subsection will generate valid geometric formations, there are several improvements that can be made. First, depending on the followed edge restitution sequence the algorithm will often generate rule sets with ambiguous witness functions. For instance, the first two rules of the set (3) have the same witness, yet different actions. Since only one rule can be applied at any given encounter of two robots the problem of rule selection has been introduced. Second, the rule set constructed by the algorithm represents a single path (or trajectory) from the initial state of an unorganized group of robots to the goal state of a valid and completed formation. Although many robot encounters will occur over the course of the assembly process, only a specific set of encounters in the proper order guarantees that the goal state will be reached. This situation results in a problem of convergence. Authors of [3] and [4], argue that no binary grammar can generate a unique stable assembly because the right hand side of the rules has no cycles. They suggest then the synthesis of graph grammars with ternary rules to solve this second problem, because they are cyclic rules. In [5], the author uses binary rules only and suggests the following to solve both problems: to compute as many constructive rule sets as possible, so as to cover all the possible and random encounters of robots; to merge these sets into a single set with coherent labeling; to add a set of destructive rules; and to add any necessary mixed rules.

To satisfy the suggestions of [5], in the approach followed here all the possible sets of binary constructive rules are synthesized by making edge-restitutions for all the possible permutations. Then, the repeated sets (those corresponding to isomorphic final graphs) are removed. After that, and recalling the computed permutations, all the sets
are merged in a sequential manner and the repeated rules are removed, even if the geometric constraints are different. Clearly, a new and coherent labelling function needs to be formulated in order to preserve each vertex’s role, and in the case of tails their constraints (since the number of sets can grow very fast with the number of edges, a larger alphabet is used, \( \Sigma_t = \{ a, ab, ac, \ldots, az, ba, bb, \ldots, bz, \ldots \} \). Notice that with this alphabet a singleton robot is now represented by \( aa \) instead of simply \( a \). The rule set (4) is a complete set of constructive rules for the wedge formation presented in the previous subsection.

\[
\Phi = \begin{cases} 
     a a a a \rightarrow ab - ac A, & a g a a \rightarrow ab - ai B \\
     a a a a \rightarrow aj - ak B, & a j a b \rightarrow av - aw A \\
     a a a b \rightarrow an - ao A, & a k a a \rightarrow ax - ay B \\
     a a a j \rightarrow az - ba B, & a n a a \rightarrow ap - aq B \\
     a b a a \rightarrow af - ag B, & a n a j \rightarrow at - au B \\
     a b a j \rightarrow al - am B, & a q a a \rightarrow ar - as B \\
     a c a a \rightarrow ad - ac A, & a z a b \rightarrow bb - bc A 
\end{cases} \quad (4)
\]

Destructive rules are also necessary to avoid a formation from arriving at an unstable state, caused by some sort of failure, for instance at the sensing or communication levels. In such situation they give the rule set the necessary cyclic characteristic to improve the convergence. The obvious destructive rules are those that invert the action of the constructive ones. The extra destructive rules arise from nodes that are labeled more than once during the assembly phase.

Finally, the mixed rules, which never create or destroy a geometric constraint, allow the assembly of a formation proceed when no more applicable constructive rules are available. Since these rules are not always necessary (but must be added when they are), they were not considered in the synthesis process.

IV. Simulations

The objective here is to present and discuss a simple and intuitive example where a binary graph grammar is used. The problem to solve can be stated as follows: given three mobile robots, make them park in a line formation with inter-distances of 5m, using only local information, and assuming that a reference robot is always stopped and that the formation leader must reach the most right position.

The formation can be perfectly represented by a simple directed spanning tree with three vertices and two ordered edges of length two (scaled by 2.5 in the workspace). Since this formation graph is very simple there are only two possible permutations to restitute the edges, figures 4(a) and 4(b). Two unique sets of constructive binary rules, where \( a a a a \rightarrow ab - ac \) is common to both, are returned in an intermediate result. But, the final result of the synthesis process gives the binary graph grammar (5) (since the two unique constraints are equal, they are omitted for simplicity). Notice that the destructive rules are simply those that invert the action of the constructive ones, and that the mixed rules were intuitively added.

\[
\Phi = \begin{cases} 
     a a a a \rightarrow ab - ac, & af - ag \rightarrow a c a a \\
     a a a b \rightarrow ad - ac, & ab - af \rightarrow ad - af \\
     a c a a \rightarrow af - ag, & ab - ag \rightarrow ab - ac \\
     ab - ac \rightarrow a a a a, & ad - ac \rightarrow ab - ac \\
     ad - ac \rightarrow a a a b, & ae - ac \rightarrow af - ag 
\end{cases} \quad (5)
\]

All the possible state transitions may be modelled as a finite state automaton ruled by the grammar, figure 5. The initial state corresponds to an unorganized group of singletons, and the marked ones correspond to the desired completed formation. Notice that the marked state \( ad - af - ag \) may only be reached with the use of mixed rules. To interpret the grammar three mobile robots with equal characteristics were considered. The robot \( i \) has a configuration space \( [x_i, y_i, \theta_i]^T \), a differential drive model, and has the linear velocity \( v_i \) and the angular velocity \( \omega_i \) as control inputs. The kinematic model describing the motion can be expressed as

\[
\begin{align*}
    \dot{x}_i &= v_i \cos \theta_i \\
    \dot{y}_i &= v_i \sin \theta_i \\
    \dot{\theta}_i &= \omega_i
\end{align*} \quad (6)
\]

In order to make the robot \( i \) go to a specified goal the closed-loop feedback control laws were implemented as

\[
\begin{align*}
    v_i &= k_1 \rho_i \\
    \omega_i &= k_2 (s_i - \theta_i) + k_3 s_i
\end{align*} \quad (7)
\]

where: \( k_1, k_2 \) and \( k_3 \) are proportional gain parameters; \( \rho \) is the distance to a goal; and \( s \) is the slope from the geometric center to a goal.
It is assumed that the robots can only communicate with each other within a 5-meter radius. Also, one robot is considered to be always stopped with pose \([0, -20, 0]\), and the initial goal of the remaining robots is at position \((0, -20)\). The system behavior was simulated in Matlab environment for two distinct situations, and the results are shown in figure 6.

The three robots are all initialized to the singleton state: the robot which is stopped (robotA), and the robots whose trajectories are represented by dashed and solid lines (robotD and robotS, respectively). The initial pose of robotS is \([0, 20, -\pi/2]\) and the initial pose of robotD is \([10, -25, -\pi]\) for the situation of figure 6(a) and \([-2.5, 16, -2\pi/3]\) for the situation of figure 6(b). Notice that a circle representing the communication range is plotted around their positions at two different instants, the initial one, and when the late robot is sufficiently near to communicate with robotA (to see which robot comes first to negotiate with robotA).

In the situation of figure 6(a) the state of the formation starts to change by the negotiation between robotD and robotA. By the first rule they fuse into a leader-follower pair \(ab - ac\) and robotD, which has become a leader by chance, decides to stop at position \((5, -20)\) to satisfy the geometric constraint of robotA. The second state transition occurs by the third rule and robotA and robotS become a leader-follower pair \(af - ag\). When robotS arrives at position \((-5, -20)\) its geometric constraint is satisfied, the marked state \(ab - af - ag\) is reached, and the formation is completed.

In the situation of figure 6(b) the moving singletons become initially a leader-follower pair \(ab - ac\) by the first rule, and robotS starts to follow its leader towards the initial goal. The marked state \(ad - ac - ac\) is reached and the formation is completed after robotD and robotA have become a pair \(ad - ac\) by the second rule.

Due to the simplicity of the example, only the first two constructive rules were used.

V. Conclusions and Future Work

An algorithm to synthesize sets of binary constructive and destructive rules for any geometric formation definable by a directed spanning tree was implemented. The mixed rules are used only when necessary, and were not considered in the synthesis process. Based on all these rules a formation is able to grow in a bottom-up manner using a kind of negotiation via a simple broadcast protocol. This approach has several merits. Although there is a leader robot, any robot can perform this role. In order for a robot to perform any and all roles, each controller must be identical and have access to all behaviors. The approach requires each robot to sense only its nearest neighbor and no global sensing is required. Also, there are no predefined orderings or predetermined organizational strategies.

There are many aspects of this work that may be explored in the future. For instance, at the synthesis level, it would be useful to develop and implement a systematic way to generate mixed rules. In what concerns to the interpretation of the grammar, advanced controllers should be considered in order to guarantee more efficiently the geometric constraints between leaders and followers. Also, it would be interesting to examine how a given formation, namely with a large number of elements, would behave in an dynamic environment with obstacles. The switching between different types of geometric formations is certainly a possible extension to this work.

References