Synthesizing Rulesets for Programmable Robotic Self-Assembly: A Case Study using Floating Miniaturized Robots

Bahar Haghighat, Brice Platerrier, Loic Waegeli, and Alcherio Martinoli

Distributed Intelligent Systems and Algorithms Laboratory
School of Architecture, Civil and Environmental Engineering
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
{firstname.lastname}@epfl.ch

Abstract. Programmable stochastic self-assembly of modular robots provides promising means to formation of structures at different scales. Formalisms based on graph grammars and rule-based approaches have been previously published for controlling the self-assembly process. While several rule-synthesis algorithms have been proposed, formal synthesis of rulesets has only been shown for self-assembly of abstract graphs. Rules deployed on robotic modules are typically tuned starting from their abstract graph counterparts or designed manually. In this work, we extend the graph grammar formalism and propose a new encoding of the robots internal states. This allows formulating formal methods capable of automatically deriving the rules based on the morphology of the robots, in particular the number of connectors. The derived rules are directly applicable to robotic modules with no further tuning. In addition, our method allows for a reduced complexity in the rulesets. In order to illustrate the application of our method, we extend two synthesis algorithms from the literature, namely Singleton and Linchpin, to synthesize rules applicable to our floating robots. A microscopic simulation framework is developed to study the performance and transient behavior of the two algorithms. Finally, employing the generated rulesets, we conduct experiments with our robotic platform to demonstrate several assemblies.

1 Introduction

Self-assembly (SA) plays a key role in many of the natural structuring phenomena at all scales. SA is defined as the reversible and spontaneous phenomenon of an ordered spatial structure emerging from the aggregate behavior of simpler preexisting entities, through inherently local and random interactions in the system. In recent years, SA has been extensively studied both as an enabling technique for micro/nano-fabrication, and as a coordination mechanism for distributed robotic systems of miniaturized modules with limited capabilities, where highly stochastic sensing, actuation, and interactions are inevitable.

SA has been used as a means for coordination in several robotic systems [11], [12], [2], [8]. Programmable SA has been demonstrated in [11], [8]. In [11],

a deterministic and quasiserial approach to shape formation is implemented in a large swarm of miniaturized Kilobot robots. Stochastic SA can be realized by taking advantage of the stochastic ambient dynamics for module transportation. In [8], the robots stochastically self-assemble on an air table based on their internal rule-based behavior. Using a synthesis algorithm, a ruleset is first derived for SA of a similar abstract target graph. The rules are then tuned to suit the specific morphology of the robots.

The problem of ruleset synthesis for programmable SA of graphs was first addressed in [9]. In [10], the formalism of graph grammars is applied to the SA of graphs and two rule synthesis algorithms are presented. A deadlock situation is discussed, where the number of copies of the target being built in parallel is higher than the maximum feasible number, considering the total number of available agents. In the same work, in order to avoid deadlocks the authors propose a disassociation rule which requires implementation of a consensus algorithm among the agents. In [3], the formalism of graph grammars is employed and it is shown that SA of graphs can be achieved while avoiding deadlocks by introducing probabilistic dissociating rules. Two formal rule-synthesis algorithms, Singleton and Linchpin, are introduced in the same work. In [4], weighted graphs are considered in a case study to encode the geometric orientations of the edges. Stochastic SA of simulated underwater robots in 3D using a rule-based approach has been studied in [5]. A rule-based approach incorporating a state machine is manually designed for the specific simulated platform and targets under study.

While several formal rule-synthesis algorithms have been proposed for programmable SA of graphs, the derived rules are not directly applicable to robotic SA where orientation of the bonding links determines the structure and is part of the internal state of the modules. In this work, we extend the graph grammars formalism for the problem of ruleset synthesis for programmable SA of robots. In particular, we extend the concept of abstract graphs by introducing vertices with link-slots and propose a new way to encode a robot's internal state. This allows formulating general methods for synthesizing rules directly applicable to robotic modules. Each module is associated with one vertex in the graph and its internal state is encoded by a control state label and an orientation index. For a module with N connectors, we achieve a ruleset complexity of $O(N)$ compared with $O(N^2)$ obtained in [8]. Using our method, we extend the Singleton and Linchpin algorithms from [3] to synthesize rules for our miniature floating robots. These two algorithms incorporate reversible rules, a feature offering two key advantages. First, reversibility is necessary for scenarios in highly stochastic environments, where permanent bonds are not always feasible. Second, by exploiting reversible rules the system can recover from deadlock situations.

We begin in Section 2 by describing the SA process in our robotic system. In Section 3, the graph grammars formalism for SA of graphs is summarized. Section 4 discusses our proposed extension for the case of SA of robots. Section 5 describes synthesis of rules for SA of our robots using two algorithms. In Section 6, we detail the simulation framework. Simulated and experimental results are presented in Section 7, with the conclusions offered in Section 8.

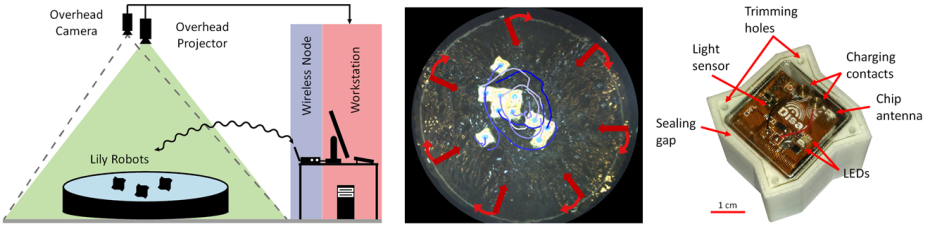

Fig. 1: An overview of the system: The experimental setup (left). Visual tracking of ten Lily robots during an experiment (middle). The Lily robot (right).

2 Fluidic Self-assembly of Lily Robots

Our system consists of two main components: 1) the Lily robots, originally presented in [7], which serve as the building blocks of the SA process, and 2) the experimental setup built around them. Lilies are endowed with four custom-designed Electro-Permanent Magnets (EPM) to latch and also to communicate locally with their neighbors. They can also communicate over a radio link to a base station to receive commands, new firmware, or to report specific information. Being power-autonomous, the robots can actively take part in the assembly process at all times. Given a target structure, an appropriate ruleset is derived as explained in Section 5, and deployed on all robots through wireless bootloading. The robots' EPM latches are by default enabled, resulting in a default latching upon meeting another robot. Once latched, the EPM-to-EPM inductive communication channel is physically established. The robots then exchange their internal states and look for an applicable rule in their ruleset. If no applicable rule is found, they unlatch by turning off their EPM latches; otherwise they remain latched and update their internal states accordingly. Each robot then updates the base station with its new internal state over the radio. Lilies are not self-locomoted, they are instead stirred by the flow field produced within a tank by several peripheral pumps. To monitor the evolution of the system, we use an overhead camera to visually track a passive marker on the top of each robot.

3 Graph Grammars for Self-Assembly of Graphs

In this section we summarize the graph grammars formalism for formulating SA of graphs as presented in [8], [3]. A labeled graph is a triple $G = (V, E, \ell)$ where $V = \{1, ..., N\}$ is the set of vertices, $E \subset V \times V$ is the set of edges, and $\ell : V \to \Sigma$ is a labeling function, with Σ being a set of labels. A pair of vertices $\{x, y\} \in E$ is represented by xy. The $n_E(k)$ represents the neighbors of vertex k relative to the edge set E. Two graphs are considered to be isomorphic when there exists a bijection $h : V_{G1} \to V_{G2}$ such that $ij \in E_{G1} \Leftrightarrow h(i)h(j) \in E_{G2}$. The function h is called a witness. A label-preserving isomorphism has the additional property that $\ell_{G1}(x) = \ell_{G2}(h(x)), \forall x \in V_{G1}$. A graph G is said to contain a graph H if a subgraph of G is isomorphic to H.

Definition: A rule is an ordered pair of graphs $r = (L, R)$ such that $V_L = V_R$. The graphs L and R are the left hand side (LHS) and right hand side (RHS) of the rule r. A binary rule can be depicted as $a \quad b \rightharpoonup c - d$, with the characters denoting the labels of the two engaged vertices.

Definition: A rule $r = (L, R)$ is applicable to a graph G if there exists $I \subset V_G$ such that the subgraph $G \subset I$ has a label-preserving isomorphism $h : I \to V_L$.

Definition: The triple (r, I, h) is called an action. Application of an action with $r = (L, R)$ to G gives a new graph $G' = (V_G, E_{G'}, l_{G'})$ defined by

$$E_{G'} = (E_G - xy : xy \in E_G \cap I \times I) \cup (xy : h(x)h(y) \in E_R)$$

$$\ell_{G'}(x) = \begin{cases} \ell_G(x), & \text{if } x \in V_G - I \\ \ell_R(h(x)), & \text{otherwise} \end{cases}$$

Definition: The complement or reverse of a rule $r = (L, R)$, is $\bar{r} = (R, L)$, such that $G \xrightarrow{r,I,h} G' \xrightarrow{\bar{r},I,h} G'' = G$.

Definition: A trajectory of a system (G_0, ϕ), where G_0 is the initial graph of the system and ϕ is a ruleset, is a finite or infinite sequence of $G_0 \xrightarrow{r_1,I,h} G_1 \xrightarrow{r_2,I,h} G_2 \xrightarrow{r_3,I,h} ...$

Given a set of rules ϕ, we can study the sequences of graphs obtained from successive application of the rules in ϕ. For a probabilistic ruleset, a probability may be associated with each rule by the mapping $P : \phi \to (0, 1]$, indicating the tendency for the corresponding event to take place provided that the conditions under which the rule is applicable are met. The formal rule-synthesis methods proposed for programmable SA of graphs automatically generate a ruleset ϕ for assembling a desired target by iteratively browsing and parsing the target graph [10], [8], [3]. Section 5 provides details on the functionality of such methods and how they can be extended to generate rules for SA of robots.

4 Graph Grammars for Self-Assembly of Robots

In this section we explain how we extend the graph grammars formalism to formulate the synthesis problem for programmable SA of robots. While the SA process in a system of atomic agents can be directly modeled by abstract graphs evolving over time, for the case of robotic modules the morphology of the robots, in particular the orientation of the links they may form, strictly determines the shape of the resulting structure. This information can not be directly encoded in the abstract graphs. Fig. 2 (left) gives a simple illustration of this issue.

The method in [8] associates each latching connector on the robot with one vertex in the graph and connects them using permanent links, as depicted in Fig. 2 (middle). This method of encoding the internal state of the robots within a graph grammar formalism has several drawbacks. First, the graph modeling the system is augmented with vertices and edges which encode redundant information, resulting in an increased complexity of model analysis and simulation. Second, manual tuning of the rules is necessary to obtain a ruleset applicable on

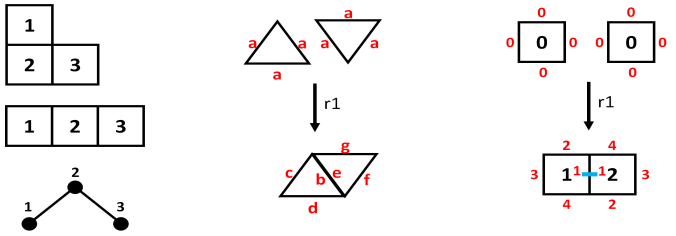

Fig. 2: Different structures with similar graph representation (left). Association of one label with one latch [8] (middle). Relative CCW hop numbering (right).

the robots. A synthesis algorithm is first run on an abstract description of the desired target, the resulting rules are then manually adapted to account for the correct orientation of the forming links. Third, for a robot with N connectors each acquiring a dedicated state label, the ruleset complexity grows in $O(N^2)$.

Our goal is to be able to formulate general methods for automatic synthesis of rules for programmable SA of robots. To this end, we extend the notion of labeled graphs by extending the definition of vertices and labels. While we are particularly interested in scenarios involving our Lily robots in 2D, the assumptions we make are general enough to be directly applied to similar platforms. The method is also easily applicable to 3D SA with similar assumptions.

Definition: An extended vertex has ordered link-slots which correspond to the latching connectors of a robotic module. The numbering on the slots is assumed to match the one of the robot, following a counter-clockwise (CCW) rotation convention. We assume that the robotic modules have a rotational symmetry. As a result, for an isolated module the connectors are anonymous.

Definition: An extended label is a pair $l = (l_a, l_n)$ encoding the internal state of a module. l_a represents the control state of the robotic module and l_n represents the index of the most recently engaged connector.

Definition: An extended labeled graph is a quadruple $G = (V, E, S, \ell)$ where $V = \{1, ..., N\}$ is the set of extended vertices, $E \subset V \times V$ is the set of edges, $S : E \to K \times K$ defines which slots are involved in a link between two vertices, and $\ell : V \to \Sigma$ is a labeling function, with Σ being a set of extended labels.

Following the extension of the graphs, the rules are also extended to be described using elements which are a combination of a control state variable and a relative hop number. The idea is that a robotic module can only take part in a reaction defined by a certain rule if it has the appropriate control state and is participating in the reaction with the appropriate orientation. We assume that the robotic modules exchange information of their respective internal states once their connectors are latched. More specifically, once one of the connectors is engaged, the robot may communicate its internal state in the form of a relative extended label of $l = (l_a, l_h)$ with l_a being the robot's control state and l_h being a relative hop number which represents the relative orientation of the currently engaged connector with respect to its predecessor, assuming a CCW

hop convention (see Fig. 2, right). For a vertex with an extended label of (l_a, l_n) on a robot with N connectors $l_h = [(l_n - l_c) mod N] + 1$, where l_n and l_c are the indexes of the most recently and the currently engaged connectors, respectively.

Definition: An extended rule is an ordered pair of extended graphs $r = (L, R)$. An extended binary rule can be depicted as $l_1 \quad l_2 \rightharpoonup l_3 - l_4$, with the $l_i = (l_{ia}, l_{ih})$ values being the relative extended label of the engaged vertex.

5 Synthesizing Rules for Robots

In the previous section, we explained the extension of the graph grammars formalism to the case of SA of robots. Our goal is to employ the extended formalism to 1) automatically synthesize rules to control programmable SA of robotic modules, and 2) model and simulate the evolution of the SA process in a system of robotic modules. Here we explain how our extended formalism may be used to formulate formal methods for deriving rules for SA of robots of arbitrary shapes, for a given target. In particular, we pick two synthesis algorithms from the literature, namely Singleton and Linchpin, presented in [3], which are capable of deriving rules for SA of graphs for any given acyclic target. Using our formalism, we extend these algorithms such that they generate rules for programmable SA of robots, for a given target represented as an extended graph. We then use the extended Singleton and Linchpin to synthesize rulesets for SA of our Lily robots for two specific targets, a chain and a cross structure, consisting of 6 robots.

For a given target graph G, *Singleton* generates a serial ruleset where each rule progresses the SA of the target graph by appending an isolated vertex to the structure. In contrast, *Linchpin* synthesizes a parallel ruleset, where the target graph is assembled from each leaf towards a final vertex, with the process culminating in two subgraphs joining together [3]. As an example consider $G = (V = \{1, 2, 3, 4, 5, 6\}, E = \{12, 23, 34, 45, 56\})$, assuming vertex 2 as the root vertex fed to the algorithms in [3], the resulting rulesets are as below:

$$\phi_{Singleton} = \begin{cases} 0 & 0 \leftrightharpoons 1 - 2 & (r1, \bar{r1}) \\ 1 & 0 \leftrightharpoons 3 - 4 & (r2, \bar{r2}) \\ 4 & 0 \leftrightharpoons 5 - 6 & (r3, \bar{r3}) \\ 6 & 0 \leftrightharpoons 7 - 8 & (r4, \bar{r4}) \\ 8 & 0 \leftrightharpoons 9 - 10 & (r5, \bar{r5}) \end{cases} \qquad \phi_{Linchpin} = \begin{cases} 0 & 0 \leftrightharpoons 1 - 2 & (r1, \bar{r1}) \\ 0 & 0 \leftrightharpoons 7 - 8 & (r2, \bar{r2}) \\ 2 & 0 \leftrightharpoons 3 - 4 & (r3, \bar{r3}) \\ 4 & 0 \leftrightharpoons 5 - 6 & (r4, \bar{r4}) \\ 8 & 6 \leftrightharpoons 9 - 10 & (r5, \bar{r5}) \end{cases}$$

5.1 Singleton and Linchpin for Robots

Alg. 1 depicts the pseudo codes for the extended Singleton algorithm for robots with N connectors, denoted as SingletonR, and the original one for abstract graphs, denoted as SingletonG. l, k, and $N_E(k)$ denote the largest label, the root vertex, and the neighbors of node k with respect to edge set E, respectively. For a given target graph $\hat{G}$, running $SingletonG((V_{\hat{G}}, E_{\hat{G}}, k, 0))$ for any $k \in V_{\hat{G}}$ generates a ruleset. The ruleset allows the SA process to grow the target graph outwards from the starting vertex k. Similarly, SingletonR generates a ruleset for robots based on a given target structure, represented by an extended

```
1: C : (V, E, S, L, k, l)
2: procedure SingletonR(C)
3:     φ ← ∅
4:     if |n_E(k)| = 0 then
5:         return (l, φ)
6:     else
7:         {v_j : j = 1, 2, ..., |n_E(k)|} ← n_E(k)
8:         for j = 1 to |n_E(k)| do
9:             (s_k, s_j) ← S(v_k, v_j)
10:            l_k ← GVL(L, s_k, v_k)
11:            l_j ← GVL(L, s_j, v_j)
12:            l̄ ← IncrementState(l, 1)
13:            l ← IncrementState(l, 2)
14:            φ ← φ ∪ {l_k l_j ⇌ l̄ − l}
15:            SVL(L, v_k, s_k, l̄)
16:            SVL(L, v_j, s_j, l)
17:            Let (V^j, E^j, S^j) be the
    component of (V, E − {kv_j}) containing v_j
18:            C_j : (V^j, E^j, S, L, v_j, l)
19:            (l, φ_j) ← SingletonR(C_j)
20:            φ ← φ ∪ φ_j
21:        end for
22:    end if
23:    return (l, φ)
24: end procedure

25: procedure GVL(L, s, v)
26:    (l_a, l_n) ← L(v)
27:    l_h ← (l_n − s + 1) (mod N)
28:    return (l_a, l_h)
29: end procedure

30: procedure SVL(L, v, s, l)
31:    (l_a, l_h) ← l(1 : 2)
32:    l_n ← s
33:    L(v) ← (l_a, l_n)
34: end procedure

35: procedure IncrementState(l, k)
36:    return (l_a + k, l_n)
37: end procedure
```

```
1: C : (V, E, k, l)
2: procedure SingletonG(C)
3:     φ ← ∅
4:     if |n_E(k)| = 0 then
5:         return (l, φ)
6:     else
7:         {v_j : j = 1, 2, ..., |n_E(k)|} ← n_E(k)
8:         l̄ ← l
9:         for j = 1 to |n_E(k)| do
10:            φ ← φ ∪ {l̄ 0 ⇌ (l + 1) − (l + 2)}
11:            l̄ ← l + 1
12:            l ← l + 2
13:            Let (V^j, E^j) be the
    component of (V, E − {kv_j}) containing v_j
14:
    C_j : (V^j, E^j, v_j, l)
15:            (l_j, φ_j) ← SingletonG(C_j)
16:            φ ← φ ∪ φ_j
17:            l ← l_j
18:        end for
19:    end if
20:    return (l, φ)
21: end procedure
```

Alg. 1: SingletonR for robotic SA and SingletonG for SA of graphs [3].

graph $G = (V_G, E_G, S_G)$, where $S(v_i, v_j)$ returns the ordered pair of (s_i, s_j), the involved link-slots on the two linked vertices. $L(v)$ returns the current extended label of a vertex, (l_a, l_n). The GVL (short for Get Vertex Label) procedure returns the ordered pair of (l_a, l_h) by updating the value of l_h such that it indicates the relative position of the currently engaged slot, s, with respect to the previously engaged one. The SVL (short for Set Vertex Label) procedure updates the extended label (l_a, l_n) by updating the value of l_n considering the value of the applied label. Compared to the SingletonG algorithm where only the state labels are synthesized, SingletonR produces the relative hop number l_h indicating the proper linking orientation as well. The combination of these two values provides a general description of the full internal state of a robot. Alternatively for a robot with N connectors, the internal state may be fully described using an ordered N-tuple by associating one state label to each connector. Considering interactions between any two connectors, this would result in a ruleset complexity of $O(N^2)$.

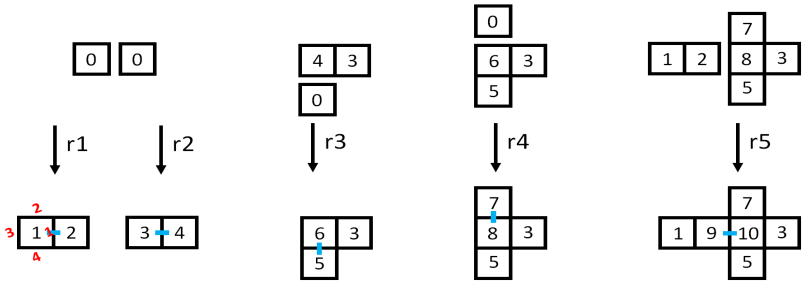

Fig. 3: Progress of the SA process for cross shape employing ϕ_L.

Utilizing our extended label convention, we obtain a ruleset complexity of $O(N)$. LinchpinR is similarly obtained by extending the standard Linchpin algorithm with the notion of link-slots. We skip its pseudo code is for brevity here.

5.2 Rulesets for Self-assembly of Lily Robots

The rulesets returned by SingletonR for a chain, ϕ_S, and the one returned by LinchpinR for a cross, ϕ_L, using 6 Lilies are reported below. The (l_a, l_h) notation is used for the relative extended labels and the reverse rules are separated.

$$\phi_S = \begin{cases} (0,0)\ (0,0) \xrightarrow{r1} (1,1)-(2,1) \\ (1,3)\ (0,0) \xrightarrow{r2} (3,1)-(4,1) \\ (4,3)\ (0,0) \xrightarrow{r3} (5,1)-(6,1) \\ (6,3)\ (0,0) \xrightarrow{r4} (7,1)-(8,1) \\ (8,3)\ (0,0) \xrightarrow{r5} (9,1)-(10,1) \\ (1,1)-(2,1) \xrightarrow{\bar{r1}} (0,0)\ (0,0) \\ (3,1)-(4,1) \xrightarrow{\bar{r2}} (1,3)\ (0,0) \\ (5,1)-(6,1) \xrightarrow{\bar{r3}} (4,3)\ (0,0) \\ (7,1)-(8,1) \xrightarrow{\bar{r4}} (6,3)\ (0,0) \\ (9,1)-(10,1) \xrightarrow{\bar{r5}} (8,3)\ (0,0) \end{cases} \quad \phi_L = \begin{cases} (0,0)\ (0,0) \xrightarrow{r1} (1,1)-(2,1) \\ (0,0)\ (0,0) \xrightarrow{r2} (3,1)-(4,1) \\ (0,0)\ (4,4) \xrightarrow{r3} (5,1)-(6,1) \\ (0,0)\ (6,3) \xrightarrow{r4} (7,1)-(8,1) \\ (2,3)\ (8,2) \xrightarrow{r5} (9,1)-(10,1) \\ (1,1)-(2,1) \xrightarrow{\bar{r1}} (0,0)\ (0,0) \\ (3,1)-(4,1) \xrightarrow{\bar{r2}} (0,0)\ (0,0) \\ (5,1)-(6,1) \xrightarrow{\bar{r3}} (0,0)\ (4,2) \\ (7,1)-(8,1) \xrightarrow{\bar{r4}} (0,0)\ (6,3) \\ (9,1)-(10,1) \xrightarrow{\bar{r5}} (2,3)\ (8,4) \end{cases}$$

Consider ϕ_S. While the state labels returned by SingletonR for Lilies are similar to the ones for a chain shape in abstract graphs, it can be seen that the values of $l_h = 3$ on the LHS of the rules dictate 2 hops between the successive latching events, resulting in a linear structure considering the square shaped modules. The reverse rules all have $l_h = 1$ at the LHS, indicating that the rule happens at the slot engaged the latest. Fig. 3 depicts the SA process employing ϕ_L on 6 Lilies. Each square represents a Lily, labeled with the l_a value. The most recent engaged link-slot is indicated with a blue mark, while the relative hop numbers of l_h are marked in red for one Lily. For each Lily, numbering the slots always starts with $l_h = 1$ at the most recently engaged slot and follows a CCW convention. Note that the synthesis algorithms only generate the rules; appropriate probabilities should be associated with forward and reverse rules to allow the system to recover from deadlocks, while reliably forming the target.

6 Simulation Framework

In order to compare the performance of our rulesets for SA of robots and to study the transient behavior, we develop a microscopic simulation framework. Our approach is based upon the abstract model for randomized interactions among atomic agents introduced in [3]. We build on this method in two ways. First, in order to model interactions between robots the notion of extended graphs along with appropriate geometrical constraints is utilized. Second, we introduce a new shape recognition method which is an extension over a graph isomorphism check to track the progress of the SA process in the system.

6.1 Random Pairwise Interactions

In our extended formalism, a random pairwise interaction dynamics is defined as a quadruple (G, F, ϕ, P). Rule probabilities are assigned by $P : \phi \rightarrow (0, 1]$. The set of pairs of disjoint vertices is defined as $PW(G) = \{(x, y) :/\exists I \subset G | (x, y) \in V_I, x \neq y\}$, where I is a connected subgraph of G. The set $PW(G)$ defines modules among which an interaction is feasible as they are not on the same sub-assembly. $F(G)$ maps an extended graph G to probabilities of pairwise vertex selections from V_G. A random trajectory of the system, is generated by sampling $F(G_t)$ at each time instant to obtain a pair (x, y) and then executing an appropriate action on the selected pair. For two selected vertices to interact, the link-slots are chosen randomly from the available slots. Sampling from $F(G_t)$ introduces an inherent stochasticity to the trajectories even if the ruleset contains only deterministic rules. The interaction probabilities, defined by $F(G_t)$, depend on the current graph G_t and can be calibrated.

6.2 Shape Recognition

Tracking the progress of the SA process of the simulated system requires a mapping between the connected components of the graph of the system and the shape of the corresponding sub-assemblies. For the case of SA of graphs where the system is represented by an abstract graph at each time instant, this describes a problem of graph isomorphism. However, for the case of our extended graphs, the relative position of the engaged slots need to be taken into account to recognize the shapes. We propose a simple method for recognizing the shapes based on traversing the connected components of the extended graph and constructing a series of locations of the Center Of Mass (COM) of the robotic modules. The relative ordering of the slots of neighboring modules determines the orientation of each traverse. The series of locations are then rotated and translated such that all coordinates are positive. The resulting ordered set is used as the identifier of the structure. This method can be applied to modules with a variety of shapes. Our method is sufficient for the case of structures confined in 2D and is substantially less computationally expensive than general approaches [1], [6].

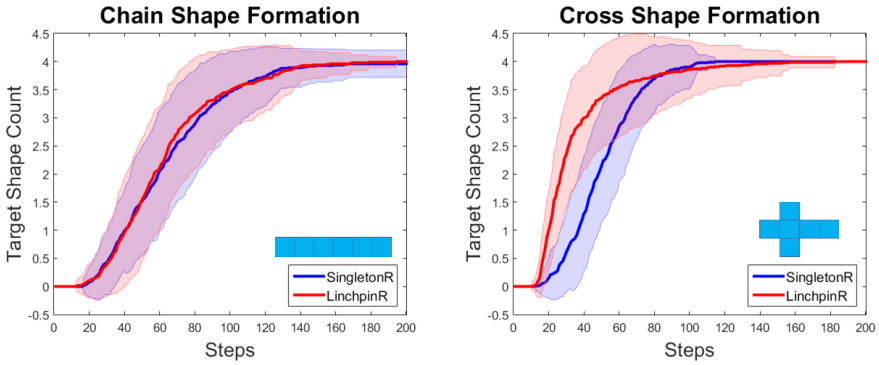

Fig. 4: Comparison of rulesets derived by the two extended synthesis algorithms for the two target structures of chain and cross shape. The solid lines and shaded regions summarize the mean and standard deviation of 100 runs, respectively.

7 Experiments and Results

SingletonR and LinchpinR algorithms are utilized to derive rules for 1) a target assembly of a chain shape, and 2) a target assembly of a cross shape, both of size 6. The resulting rulesets are deployed in microscopic simulations with 24 modules, and on 6 real robots. Within a ruleset, all the rules with identical LHS are set to be equi-probable. For forward rules $P(.) = 1$ and for reverse rules $P(.) = 0.1$ is chosen. The finishing rule is chosen to be irreversible in all the rulesets, giving rise to stable target assemblies once they are formed. Care should be taken in comparing simulated and experimental results. First, a higher number of available modules compared to the target size in simulation allows for inherently larger opportunities for interaction, particularly in the early stages of the process. Second, the simulation framework assumes all interactions to be equiprobable, this can be a good assumption when the number of available modules is much higher than the target size. Third, the simulation results are reported as a function of steps, representing formation events in the system, a progress unit suitable for measuring the concurrency of the rulesets.

Fig. 4 shows the performance of the rulesets derived by the two extended synthesis algorithms for the two target assemblies in simulation. The vertical axis shows the number of copies of the target assembly in the system at each step. For

Table 1: Formation Time Statistics

Algorithm	Target	Mean (s)	Std. (s)
SingletonR	Chain shape	844.3	165.8
LinchpinR	Chain shape	941.1	138.9
SingletonR	Cross shape	181.1	25.8
LinchpinR	Cross shape	190.3	77.6

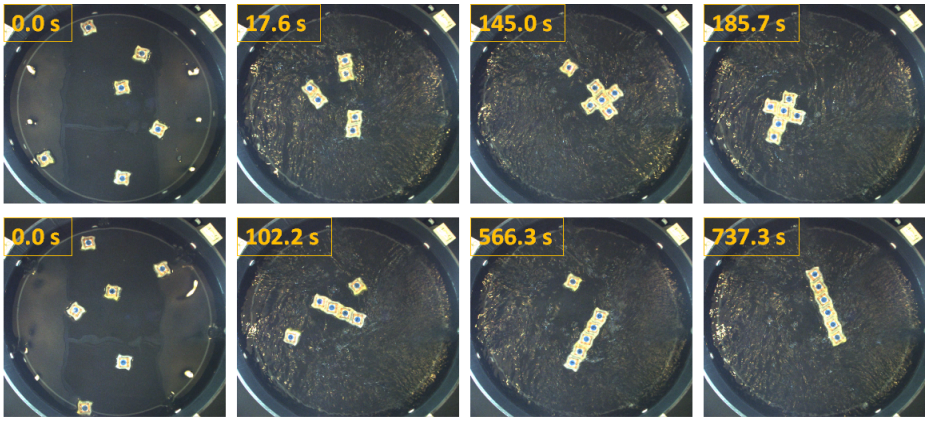

Fig. 5: SingletonR rulesets for chain and cross shapes on six Lily robots.

the cross shape target, the naturally serial ruleset of SingletonR is outperformed by the more concurrent one of LinchpinR. For the chain shape target, the rulesets of the two algorithms perform similarly. Looking into the generated rules, LinchpinR builds dimers with two possible labeling assigned probabilistically, while SingletonR adds modules one by one, labeled deterministically.

For the experimental studies, 6 Lily robots were programmed with the derived rulesets to build the two target structures. Each experiment was repeated five times. Table 1 details the formation time statistics. Considering the chain shape, while the average formation time for SingletonR is less than that of LinchpinR, SingletonR exhibits a higher standard deviation. This can be ascribed to the assembly strategy of the rulesets. LinchpinR builds the target out of dimers and requires two dimers labeled differently. Since the labeling is done at random, when the available modules are scarce this can easily result in longer formation times. More generally, LinchpinR does not necessarily make the best use of the available resources. For the cross shape, both the smallest and the largest formation times were obtained by LinchpinR. This can be explained by considering the interaction between the intermediate sub-assemblies. While LinchpinR builds the target through four concurrent steps as opposed to SingletonR's five, the relative orientation of the connecting sub-assemblies is more easily achieved for SingletonR where one component, i.e. the isolated Lily, is always symmetric.

8 Conclusion

In this paper, we addressed the problem of rule synthesis for programmable SA of robots. We extended the graph grammar formalism to account for the morphology of the robots and proposed a formal method to automatically synthesize rules for robots. We introduced the notion of extended graphs comprising vertices with ordered link-slots representing the robotic modules' connectors. The

state of each module is represented by an extended label. We showed that our formalism achieves a ruleset complexity of $O(N)$ compared to the conventional methods' $O(N^2)$. Using our method, two synthesis algorithms originally introduced for SA of graphs were then extended to synthesize rules for our robots. Studies on the synthesized rulesets in simulation and real experiments demonstrated the functionality of our method. In the future, we will investigate novel rule-synthesis algorithms allowing for higher concurrency in the process by considering geometrical features of the target. Finally, we plan to fully utilize our setup to conduct systematic real experiments involving up to 100 Lily robots.

Acknowledgments.

This work has been sponsored by the Swiss National Science Foundation under the grant numbers 200021_137838/1 and 200020_157191/1.

References

1. Asadpour, M., Ashtiani, M.H.Z., Sproewitz, A., Ijspeert, A.: Graph signature for self-reconfiguration planning of modules with symmetry. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5295–5300 (2009)
2. Ayanian, N., White, P.J., Hálász, A., Yim, M., Kumar, V.: Stochastic control for self-assembly of xbots. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. pp. 1169–1176 (2008)
3. Fox, M., Shamma, J.: Probabilistic performance guarantees for distributed self-assembly. IEEE Transactions on Automatic Control 60(12), 3180–3194 (2015)
4. Fox, M.J., Shamma, J.S.: Communication, convergence, and stochastic stability in self-assembly. In: IEEE International Conference on Decision and Control. pp. 7245–7250 (2010)
5. Ganesan, V., Chitre, M.: On stochastic self-assembly of underwater robots. IEEE Robotics and Automation Letters 1(1), 251–258 (2016)
6. Golestan, K., Asadpour, M., Moradi, H.: A new graph signature calculation method based on power centrality for modular robots. In: Distributed Autonomous Robotic Systems, pp. 505–516 (2013)
7. Haghighat, B., Droz, E., Martinoli, A.: Lily: A miniature floating robotic platform for programmable stochastic self-assembly. In: IEEE International Conference on Robotics and Automation. pp. 1941–1948 (2015)
8. Klavins, E.: Programmable self-assembly. IEEE Control Systems 27(4), 43–56 (2007)
9. Klavins, E.: Automatic synthesis of controllers for distributed assembly and formation forming. In: IEEE International Conference on Robotics and Automation. pp. 3296–3302 (2002)
10. Klavins, E., Ghrist, R., Lipsky, D.: A grammatical approach to self-organizing robotic systems. IEEE Transactions on Automatic Control 51(6), 949–962 (2006)
11. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science 345(6198), 795–799 (2014)
12. Salemi, B., Moll, M., Shen, W.M.: Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3636–3641 (2006)