A Rule Synthesis Algorithm for Programmable Stochastic Self-Assembly of Robotic Modules

Bahar Haghighat and Alcherio Martinoli

Abstract Programmable self-assembly of modular robots offers promising means for structure formation at different scales. Rule-based approaches have been previously employed for distributed control of stochastic self-assembly processes. The assembly rate in the process directly depends on the concurrency level induced by the employed ruleset, i.e. the number of concurrent steps necessary to build one instance of the target structure. Our aim here is to design a formal synthesis algorithm to automatically derive rulesets of high concurrency for a given target structure composed of robotic modules. In the literature, self-assembly of (simulated or real) robotic modules has been realized through manually designed rulesets or manually adjusted rulesets generated by employing graph-grammar formalisms or metaheuristic methods. In this work, we employ an extended graph-grammar formalism, adapted for self-assembly of robotic modules, and propose a novel formal synthesis algorithm capable of generating rulesets for robotic modules by natively considering the morphology of their connectors. The synthesized rulesets induce a high level of concurrency in the self-assembly scheme by exploiting controlled information propagation, using solely local communication. Simulation results of microscopic (non-spatial) and submicroscopic (spatial) models of our robotic platform confirm higher performance of rulesets synthesized by our algorithm compared to related work in the literature.

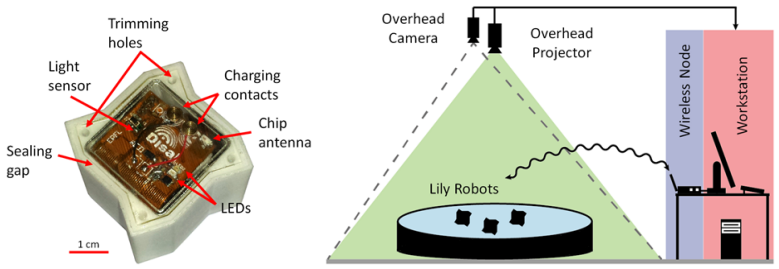
1 Introduction

Self-assembly (SA) is defined as the reversible and spontaneous phenomenon of an ordered spatial structure emerging from the aggregate behavior of simpler preex-

Bahar Haghighat and Alcherio Martinoli
École Polytechnique Fédérale de Lausanne (EPFL), Distributed Intelligent Systems and Algorithms Laboratory (DISAL), School of Architecture, Civil and Environmental Engineering, Lausanne, Switzerland, e-mail: bahar.haghighat@epfl.ch, alcherio.martinoli@epfl.ch

isting 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 [13, 1, 4]. Various implementations of SA have been demonstrated in engineered systems. For instance, a deterministic and quasi-serial approach to shape formation through programmable SA has been implemented in a large swarm of miniaturized Kilobot robots [13]. A completely different approach is to achieve SA by taking advantage of the stochastic ambient dynamics for module transportation. Such approach typically enables simpler internal design of the programmable robotic modules as well as their on-board algorithms. The robotic modules in [8] stochastically self-assemble on an air table based on their behavioral ruleset which is first derived for SA of a similar abstract target graph by a synthesis algorithm, then tuned to suit the specific modules' morphology.

The problem of ruleset synthesis for programmable SA of graphs was first addressed in [9]. In principle, all proposed formal synthesis algorithms in this context generate rules by iteratively browsing and parsing a description of the target structure. In [10], the formalism of graph-grammar is applied to the SA of graphs and two rule synthesis algorithms are proposed for acyclic and then cyclic target graphs. Both algorithms result in rulesets which build the target graph serially, i.e. by adding one atomic agent (graph vertex) at a time. The same work also discusses the deadlock situation, where the number of copies of the target being built in parallel is higher than the maximally feasible number, considering the total number of available agents. In order to avoid deadlocks the authors then propose a disassociating rule which requires implementation of a consensus algorithm among the agents. In [3, 12], SA of graphs is achieved while avoiding deadlocks by employing probabilistic dissociating rules. In order to accelerate the SA process induced by an inherently serial ruleset, the authors in [12] employ a broadcast radio communication scheme to decompose less advanced sub-assemblies in favor of the others. The formalism of graph-grammar is employed in [3] for modeling and control of SA of abstract graphs. Two formal synthesis algorithms, Singleton and Linchpin, are then introduced. While Singleton generates utterly serial rulesets, Linchpin induces a more parallel scheme and allows for sub-assemblies of bigger sizes to merge and form the target graph. Two main factors limit the concurrency induced by the Linchpin rulesets. First, in an attempt to fully avoid information propagation for scalability reasons, all sub-assemblies including the ones with identical structure are labeled distinctly, determining their final placement in the target structure. As a consequence, several rules are required to form the target, meaning more assembly time. Second, at each branching point in the target, i.e. at any vertex with two or more neighbors, the algorithm generates rules to build the branches in parallel and eventually join them, regardless of the size of the branches. This imbalance in the size of the concurrently built branches can result in unnecessary delay, as the longer branch may need more rules to form. The slow nature of stochastic SA highlights the importance of inducing a high level of concurrency in the process in order to reduce the total assembly time.



(a) (b)

Fig. 1 – (a) The Lily robot [5]. (b) Sketch of the full experimental setup [6].

In a previous contribution, we presented an extended graph-grammar formalism adapted for formulating the problem of SA of robotic modules. The formalism is based on the notion of extended vertices [7]. In addition, we showed that our extended formalism allows for a ruleset of complexity $O(N)$ compared to the $O(N^2)$ in the previous literature. In this work, we employ the extended graph-grammar formalism presented in [7], and propose a new synthesis algorithm with two key features: first, our algorithm automatically derives rules for robotic modules considering their latching connectors morphology, and second, it achieves a parallel assembly scheme by exploiting update rules (i.e. information propagation) of controllable depth between neighboring robots. In order to avoid deadlocks, our algorithm incorporates reversible rules. We consider two target structures composed of our floating Lily robots and compare the performance of rulesets derived by our algorithm and those of the extended Linchpin, adapted to generate rules for robotic modules [7], in simulation. To this end, we leverage non-spatial microscopic simulations in Matlab and high-fidelity spatial submicroscopic simulations in Webots [11].

2 Fluidic Self-assembly of Lily Robots

In this paper, we study the SA of our robotic modules in simulation. For the sake of thoroughness, here we give a brief summary of the SA process in our real world experimental system [6]. Our system consists of two main components: 1) the Lily robots, originally presented in [5], which serve as the building blocks of the SA process, and 2) the experimental setup built around them. Lilies are not self-locomoted, instead, they float on water and are stirred by the flow field produced within a tank by several peripheral pumps, colliding randomly with other robots. The robots are endowed with four custom-designed Electro-Permanent Magnets (EPM) to latch and also to communicate locally with their neighbors. Given a target structure, an appropriate ruleset is programmed on all robots. 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 applicable rules in their rule-set. If no applicable rule is found, they unlatch by switching off their EPM latches; otherwise they remain latched and update their internal states accordingly.

3 Graph Grammars for Self-Assembly of Robotic Modules

In this section, we summarize the graph-grammar formalism adapted for formulating SA of robotic modules [7].

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, $S : E \to K \times K$, with $K = \{1, ..., N_s\}$ and N_s the total number of link-slots associated with an extended vertex, defines which slots are involved in a link between two vertices. $\ell : V \to \Sigma$ is a labeling function, with Σ being a set of extended 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 extended graphs are considered to be isomorphic when there exists a bijection $h : V_{G1} \to V_{G2}$ such that $\forall ij \in E_{G1} \Leftrightarrow h(i)h(j) \in E_{G2}$. The function h is called a witness. A label-preserving isomorphism has the additional property that $\ell_{G1}(x) = \ell_{G2}(h(x)), \forall x \in V_{G1}$.

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 physical module, 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.

Once two modules are latched, each communicates its internal state in the form of a relative extended label of $l = (l_a, l_h)$ with l_a being the module's control state and l_h being a relative hop number representing the relative orientation of the currently engaged connector with respect to its predecessor, assuming a CCW hop convention. For a vertex with an extended label of (l_a, l_n) on a robotic module with N connectors, the communicated l_h is $[(l_n - l_c) mod N] + 1$, where l_n and l_c are the indexes of the most recently and the currently engaged connectors, respectively. For further details on application of extended labels see [7].

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 i.

Definition: A rule $r = (L, R)$ is applicable to a graph G if there exists $I \subset V_G$ such that the subgraph $G \cap I$ 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 $G \xrightarrow{r,I,h} G' \xrightarrow{\bar{r},I,h} G'' = G$.

Note: When forming an edge, the rule is denoted as a "link rule". When severing an edge, the rule is denoted as an "unlink rule".

Definition: An update rule can be depicted as $l_1 - l_2 \rightharpoonup l_3 - l_4$; it does not create or sever an edge between two vertexes, instead it modifies their labels.

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 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 contained 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.

4 Proposed Synthesis Algorithm

Given an acyclic target structure composed of rotationally symmetrical robotic modules with any number of connectors, our proposed synthesis algorithm derives rulesets based on two principles: 1) limiting the size of the concurrently built subassemblies to a user-defined value, and 2) unifying the rules which give rise to sub-assemblies with similar structures. The algorithm comprises two stages, each realizing one of the two principles. The first stage parses a graphical description of the target, and derives a ruleset which builds the target by merging sub-assemblies with sizes no more than the user-defined value and with distinct labelings, as a result of employing distinct rules. The second stage then processes this ruleset to identify the rules producing structures with identical morphology; such rules are then merged in a single one. As a result of the second principle, i.e. unifying the rules and consequently the labelings, the rulesets need to include update rules. Consider the case where the maximum user-defined size is 2. With the rules unified properly, all dimers (sub-assemblies composed of 2 modules) are labeled similarly. As the dimers join to build the target, the labelings of both consisting modules need to be updated to reflect their placement in the forming target structure to allow for proper further reactions, completing the target eventually. In other words, the use of update rules is an alternative to building the target out of distinctly labeled sub-assemblies, as a result of being formed through distinct link rules, according to their intended placement in the target structure. Thus in general, introducing update rules into the rulesets can reduce the number of link rules necessary to build the target, at the ex-

pense of possibly increasing the total ruleset size to include several update rules. However, this can offer a significant advantage in terms of assembly time. The occurrence of the link rules is probabilistic and is determined by the stochastic nature and dynamics of the system which is relied upon to provide proper interactions in order for the SA process to progress. The occurrence rate for link rules is usually in the order of once in tens of seconds. On the other hand, update rules are purely communicational rules and do not depend on the system dynamics. Once a proper interaction has happened and two modules have bonded successfully, the occurrence of a proper update rule is solely determined by the modules communication rate, usually in the order of once in less than a second. Fewer link rules can thus significantly decrease the total assembly time. The first principle addresses the propagation delay concerns which can cause scalability issues. Limiting the size of the concurrently built sub-assemblies allows for restricting the extent by which the update rules need to propagate. Therefore, in a robotic system with measurable propagation delay and interaction rate, the update propagation depth can be set accordingly to allow for a parallel assembly scheme while minimizing possible propagation delay faults. In order to avoid deadlocks, we employ probabilistic dissociating rules. Appropriate reverse rules are generated at the end of Stage II. It should be noted that the algorithm only generates the ruleset. Appropriate probabilities should be assigned to the rules to reliably build the target while avoiding deadlocks.

4.1 Stage I: Grow Subtrees (GS)

Stage I allows for creating concurrently built sub-assemblies similar to the concurrency created by the Linchpin algorithm [3], with the additional capability to control the maximum permitted size of such sub-assemblies. The GS algorithm employed in Stage I tries to build a given target structure using as many sub-assemblies of a defined size as possible built in parallel, before trying to join them to make a bigger sub-assembly and eventually form the target. In principle, the algorithm addresses the second issue with the Linchpin algorithm (see Section 1). Linchpin generates rules to build parallel substructures for every branch split in the target recursively in order to build the target using one final finishing rule. With one finishing rule in the ruleset, it is shown in [3] that the target can be built reliably while avoiding deadlocks by having probabilisitic dissociating rules for all rules except for the finishing rule. The GS algorithm on the other hand, permits a maximum size for the concurrently built sub-assemblies and as a result may end up building the target using several concurrent finishing rules. Stage II then processes this ruleset and results in one finishing rule. Algorithm 1 shows the pseudo code of the GS algorithm. The first call to GS is by $Size = 0$. The algorithm then recursively proceeds to create extended labels and corresponding rules, moving outwards from a starting vertex k. The ruleset returned by GS for a chain structure of size 6 and maximum sub-assembly size of 2, is depicted below along with the link rules generated by Linchpin. In order to simplify the comparison, the rulesets have been designed for an abstract graph.

```
1: C : (V,E,S,L,k,l,Size,S_max)
2: procedure GS(C)
3:     φ ← ∅
4:     φ̄ ← ∅
5:     Size ← Size + 1
6:     if |n_E(k)| = 0 then
7:         return (l, φ)
8:     else
9:         {v_j : j = 1,2,...,|n_E(k)|} ← n_E(k)
10:        for j = 1 to |n_E(k)| do
11:            s_k ← S(v_k, v_j)
12:            s_j ← S(v_j, v_k)
13:            l_k ← GVL(L, s_k, v_k)
14:            if Size < S_max then
15:                l_j ← GVL(L, s_j, v_j)
16:                l̄ ← INCREMENTSTATE(l, 1)
17:                l ← INCREMENTSTATE(l, 2)
18:                φ̄ ← φ ∪ {l_k l_j ⇌ l̄ − l}
19:                SVL(L, v_k, s_k, l̄)
20:                SVL(L, v_j, s_j, l)
21:                Size_j ← Size
22:            else
23:                φ̄ ← ∅
24:                Size_j ← 0
25:            end if
26:            Let (V^j, E^j, S^j) be the
    component of (V, E − {kv_j}) containing v_j
27:            C : (V^j, E^j, S, L, v_j, l, Size_j, S_max)
28:            (l, φ_j, Size_j) ← GS(C)
29:            φ ← φ ∪ φ̄ ∪ φ_j
30:            if Size == S_max then
31:                l_j ← GVL(L, s_j, v_j)
32:                l̄ ← INCREMENTSTATE(l, 1)
33:                l ← INCREMENTSTATE(l, 2)
34:                φ ← φ ∪ {l_k l_j ⇌ l̄ − l}
35:            else
36:                Size ← Size_j
37:            end if
38:        end for
39:    end if
40:    return (l, φ, Size)
41: end procedure

42: procedure GVL(L, s, v)
43:    (l_a, l_h) ← L(v)
44:    l_h ← (l_h − s + 1) (mod N)
45:    return (l_a, l_h)
46: end procedure

47: procedure SVL(L, v, s, l)
48:    (l_a, l_h) ← l(1 : 2)
49:    l_h ← s
50:    L(v) ← (l_a, l_h)
51: end procedure

52: procedure INCREMENTSTATE(l, k)
53:    return (l_a + k, l_h)
54: end procedure
```

Algorithm 1 – Pseudo code of the GS algorithm employed in Stage I.

$$\phi_{GS} = \begin{cases} 0\ 0 \rightarrow 1-2 & (r_1) \\ 0\ 0 \rightarrow 3-4 & (r_2) \\ 0\ 0 \rightarrow 5-6 & (r_3) \\ 4\ 5 \rightarrow 7-8 & (r_4) \\ 2\ 3 \rightarrow 9-10 & (r_5) \end{cases} \qquad \phi_{Linchpin} = \begin{cases} 0\ 0 \rightarrow 1-2 & (r1) \\ 0\ 0 \rightarrow 3-4 & (r2) \\ 0\ 2 \rightarrow 5-6 & (r3) \\ 0\ 4 \rightarrow 7-8 & (r4) \\ 5\ 7 \rightarrow 9-10 & (r5) \end{cases}$$

Consider a set of initially isolated atomic agents, all labeled 0. The rules generated by GS allow for the target to be built in two concurrent steps, i.e. first (r_1, r_2, r_3) and then (r_4, r_5), while the rules synthesized by Linchpin require three concurrent steps, i.e. first (r_1, r_2), then (r_3, r_4), and eventually (r_5).

4.2 Stage II: ReGroup Subtrees (RGS)

Stage II processes the ruleset generated by Stage I to unify the link rules which create up to the maximum size sub-assemblies and add proper update rules. The key idea of processing is to apply the rules synthesized by GS to two graphs with initially fully isolated vertices. The two graphs evolve identically in structure but differ in labeling, one graph is labeled according to the original ruleset, while in the other graph the forming sub-assemblies are processed to identify the rules with

products of identical shapes. In order to identify structures with identical shapes the shape recognition algorithm explained in Section 5.1.2 is utilized. We omit the pseudo code for the RGS algorithm employed in this stage for brevity. The RGS algorithm can be explained in four phases:

Forming dimers Unify the rules (of Stage I) which form dimers.

Forming larger sub-assemblies Grow on the dimers. Recognize the shape of the resulting sub-assemblies. Unify the rules producing identical structures.

Relabeling max-size sub-assemblies Create update rules for relabeling all the modules in sub-assemblies of up to the max-size (i.e. user-defined value) size.

Growing on max-size sub-assemblies Create necessary rules, both link and update, to form the target assembly out of the max-size sub-assemblies.

Considering the target shape of a chain of size 6 for an abstract graph, the rules generated by RGS are as below:

$$\phi_{RGS} = \begin{cases} 0 \quad 0 \quad \rightarrow \quad 1-2 & (r_1) \\ 1 \quad 2 \quad \rightarrow \quad 4-3 & (r_2) \\ 1 \quad 8 \quad \rightarrow \quad 10-9 & (r_3) \\ 1-3 \quad \rightarrow \quad 6-5 & (r_4^u) \\ 2-4 \quad \rightarrow \quad 8-7 & (r_5^u) \\ 7-9 \quad \rightarrow \quad 12-11 & (r_6^u) \\ 2-10 \quad \rightarrow \quad 14-13 & (r_7^u) \end{cases}$$

Two points are noteworthy here. First, the existence of the update rules in the ruleset (r_4^u to r_7^u). And second, the number of concurrent steps necessary for forming the target being equal to 3. Assuming that the update events are instantaneous and that the number of available modules is limited, RGS can on average build the target faster than Linchpin with the same number of necessary concurrent steps (see Section 6). This is due to the fact that RGS makes a better use of the available modules by limiting the number of distinctly labeled sub-assemblies with identical shapes. At the end of Stage II, the ruleset is augmented with proper reverse rules accounting for both the link and the update rules. The application of a reverse rule essentially takes the SA process back in time by reversing the labeling and/or the bonding.

Proposition 1: The complete ruleset ϕ_{full} generated by our proposed method for assembling a target structure described as an extended graph $G = (V, E, S, l)$ will eventually achieve the maximum possible number of copies of the target structure (i.e. maximum yield) provided that the available assembly modules executing the ruleset interact often enough and that the corresponding execution probability is set to $p = 1$ for link and update rules and to $p < 1$ for reverse rules.

Proof: The ruleset ϕ_{full} contains an unlink rule for each link and update rule. Only the last link rule has no corresponding reversal rule. Therefore, while all partially formed structures dis-assemble with a non-zero probability, the finishing rule is reversed with zero probability, therefore leading to a stable target structure. □

Proposition 2: The complete ruleset ϕ_{full} generated by our proposed method for assembling a target structure described as an extended graph $G = (V, E, S, l)$ will achieve an assembly rate at least as fast as that of a ruleset derived by the Linchpin algorithm, assuming that the update rules are applied instantaneously.

Proof: Rulesets generated by Stage I have as many link rules as the ones generated by the Linchpin algorithm, i.e. the number of edges in the target graph, as a result of forming the target out of uniquely labeled sub-assemblies. As a result of merging the link rules in Stage II, ϕ_{full} contains at most as many link rules as the ruleset ϕ_{GS} created in Stage I. Therefore, ϕ_{full} achieves the target structure in the same or fewer concurrent steps than rulesets derived by Linchpin. □

4.3 Synthesized Rulesets for Lily Robots

We consider two targets, a chain and a cross structure, each composed of six Lily robots (see Fig. 3). The rulesets returned by our algorithm (with the maximum user-defined size set to 2) for the chain structure ϕ_-, and for the cross structure ϕ_+, are reported below. The (l_a, l_h) notation is used for the relative extended labels and the reverse rules are separated. Note that the reverse rules do not correspond to a single link or update rule, but rather have a time reversal effect, taking the labeling back in time. For the sake of brevity we skip functional explanation of these rulesets, further details can be found in [7].

$$\phi_- = \begin{cases} (0,0) \quad (0,0) \quad \xrightarrow{r_1} \quad (1,1)-(2,1) \\ (1,3) \quad (2,3) \quad \xrightarrow{r_2} \quad (4,1)-(3,1) \\ (1,3) \quad (8,3) \quad \xrightarrow{r_3} \quad (10,1)-(9,1) \\ (1,1)-(3,3) \quad \xrightarrow{r_4^u} \quad (6,1)-(5,1) \\ (2,1)-(4,3) \quad \xrightarrow{r_5^u} \quad (8,1)-(7,1) \\ (7,1)-(9,3) \quad \xrightarrow{r_6^u} \quad (12,1)-(11,1) \\ (2,1)-(10,3) \quad \xrightarrow{r_7^u} \quad (14,1)-(13,1) \\ (1,1)-(2,1) \quad \xrightarrow{\bar{r}_1} \quad (0,0) \quad (0,0) \\ (5,3)-(7,3) \quad \xrightarrow{\bar{r}_{2/4/5}} \quad (3,1) \quad (4,1) \\ (3,3)-(6,1) \quad \xrightarrow{\bar{r}_{2/4}} \quad (2,1)-(1,1) \\ (4,3)-(8,1) \quad \xrightarrow{\bar{r}_{2/5}} \quad (1,1)-(2,1) \end{cases} \qquad \phi_+ = \begin{cases} (0,0) \quad (0,0) \quad \xrightarrow{r_1} \quad (1,1)-(2,1) \\ (0,0) \quad (2,4) \quad \xrightarrow{r_2} \quad (4,1)-(3,1) \\ (0,0) \quad (5,2) \quad \xrightarrow{r_3} \quad (8,1)-(7,1) \\ (2,3) \quad (7,2) \quad \xrightarrow{r_4} \quad (10,1)-(9,1) \\ (1,1)-(3,2) \quad \xrightarrow{r_5^u} \quad (6,1)-(5,1) \\ (1,1)-(10,3) \quad \xrightarrow{r_6^u} \quad (12,1)-(11,1) \\ (1,1)-(2,1) \quad \xrightarrow{\bar{r}_1} \quad (0,0) \quad (0,0) \\ (4,1)-(5,4) \quad \xrightarrow{\bar{r}_2} \quad (0,0) \quad (3,1) \\ (6,1)-(3,2) \quad \xrightarrow{\bar{r}_{2/5}} \quad (1,1)-(2,1) \\ (7,1)-(8,1) \quad \xrightarrow{\bar{r}_3} \quad (5,3) \quad (0,0) \end{cases}$$

5 Modeling Levels and Simulation Frameworks

In order to compare the performance of our rulesets for SA of robots and to study the transient behavior, we conduct simulated studies using two different simulation frameworks. In the non-spatial microscopic framework implemented in Matlab, the system is modeled as an extended graph (see Section 3), with each extended vertex in the extended graph corresponding to a robotic module. In the high-fidelity sub-microscopic simulation framework realized in Webots [11], the physics of the system including the embodiment of the robotic modules, the hydrodynamic forces acting on the robots, and the software running on them are recreated.

5.1 Microscopic Model and Simulation Framework

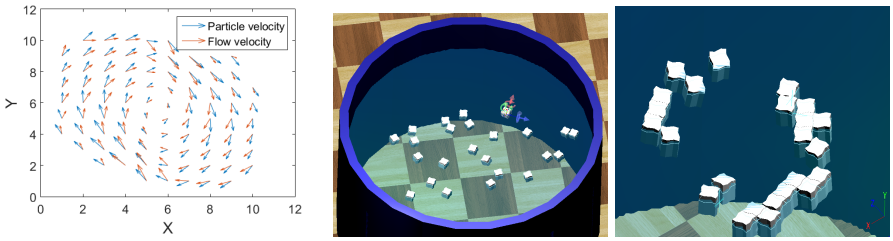
In this framework, the physical dynamics of our robotic system is abstracted into randomized interactions between atomic units in favor of gaining simulation speed. The system is represented as an extended graph which evolves over time. Each robotic modules corresponds to an extended vertex in the system graph. In order to model interactions between the robotic modules a randomized scheme along with appropriate geometrical constraints is utilized. In order to track the progress of the SA process in the system, we employ a shape recognition algorithm which is an extension over a graph isomorphism check.

5.1.1 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 $PW(G)=\{(x,y):\nexists I\subset G|(x,y)\in V_I,x\neq y\}$, 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.

5.1.2 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 employ 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 forming structures in 2D.



(a) (b) (c)

Fig. 2 – (a) Velocity field of the tracked globe and the computed fluid flow. (b) and (c) Simulated world of Lily robots in Webots.

5.2 Sub-Microscopic Model and Simulation Framework

In this context, submicroscopic means that it provides a higher level of detail than a canonical microscopic model, faithfully reproducing intra-robot details (e.g., body shape, individual sensors and actuators). In order to realistically recreate our self-assembling system in simulation, we use Webots [11], a physics-based robotics simulator. Webots uses the Open Dynamics Engine (ODE) for simulating rigid body dynamics. Additionally, in order to simulate specific not natively supported physics, it is possible to employ custom designed physics plugins. The Lily modules' CAD design as well as the robots' controller software is imported into the Webots simulated world. The rulesets programmed on the simulated robots are also identical to the case of the microscopic simulation in the previous section. The latest version of Webots supports a basic fluid node which allows for a simple uniform stream velocity, but is not capable of simulating a complex fluidic field. We used a similar approach as [2] to reproduce the complex flow field and the hydrodynamic forces.

We record the trajectory of a single floating globe (diameter of 3 cm), roughly the same size of a Lily robot, for 3 experiments with random starting positions and duration of 10 minutes each. The globe's weight is tuned such that the submersion level is similar to that of a Lily robot (25 mm below water level). The captured velocity fields are then augmented and discretized on a regular grid of 50 cells on each side, for our water tank of 120 cm in diameter. For each cell of the grid, the observed velocity vectors are averaged and assigned as the velocity of that cell. The fluid velocity field can be computed considering the drag force. The value of the Reynolds number Re determines the flow regime and the form of the drag force:

$$Re = \frac{\rho V L}{\mu} \simeq 2000 \qquad |\overrightarrow{F}_{drag}| = \frac{1}{2}\rho A C |\overrightarrow{v}_{block} - \overrightarrow{v}_{flow}|^2 \tag{1}$$

where $\rho = 10^3\ kg/m^3$ is the density of water, $V \simeq 20\ cm/s$ the experimentally-measured mean velocity of the globe, L = 3 cm the characteristic dimension, and

$\mu = 8.90 \ . \ 10^{-4}$ Pa.s the dynamic viscosity of water. The submerged area of the globe is, $A = 7\ cm^2$ and the drag coefficient constant in all directions $C = 0.47$. The velocity and acceleration of the globe are computed using the captured trajectory data. Considering the mass of the globe m, the flow velocity is then computed as:

$$\overrightarrow{v}_{flow} = \overrightarrow{v}_{block} + \frac{m\overrightarrow{a}}{\sqrt{\frac{1}{2}\rho ACm\sqrt{a_x^2 + a_y^2}}} \tag{2}$$

We developed a physics plugin for Webots that applies the drag force to the simulated Lily module based on the velocity of the robot and the flow velocity at its location at each time instant. In order to account for rotational effects, the drag force is integrated over each face of the Robot. Each face is divided into $N = 10$ sections, and the drag force is computed for each section using Eq. 1 with C being the estimated Lily robot's drag coefficient C_{Lily}. The physics plugin also adds a stochastic force $F_s \sim \mathcal{N}(0, \sigma_f^2)$ to the center of mass of each robotic module in order to account for non-modeled effects (e.g., physical irregularities, turbulences). The values of $C_{Lily} = 0.7$ and $\sigma_f = 50$ mN were empirically set based on our previous findings in [2]. Two independent Kolmogrov-Smirnov (KS) tests showed that the simulated and real distributions of the step lengths and step angles extracted from the trajectories have a KS distance of less than 0.2.

6 Experiments and Results

We evaluate our algorithm leveraging the two modeling levels and corresponding simulation frameworks of Section 5, studying the SA process in a swarm of 24 initially isolated Lily robots. Two target shapes of a chain and a cross shape, each composed of 6 robotic modules, are considered. A maximum of 4 copies of each target can be assembled thus. The microscopic simulation framework (see Section 5.1) employs a random pairwise interaction dynamics. All interactions among microscopic nodes are set to be equiprobable, i.e. we make the assumption that the system is perfectly mixed. We employ rulesets synthesized by our algorithm (see Section 4.3) and the extended Linchpin algorithm [7], for our simulated Lily robots. For forward and update rules $P(.) = 1$ and for reverse rules $P(.) = 0.01$ is set. The finishing rule is set to be irreversible in all the rulesets, giving rise to stable target assemblies once they are formed. Figure 3 depicts the performance of the rulesets derived by our algorithm along with the ones of the extended Linchpin for the two target shapes. While for the submicroscopic simulations the results are reported as a function of the experimental time (emulating the real time progress in a real experiment), the results of the microscopic simulations are reported as a function of steps, each step representing a formation event in the system. While such choice makes the results of the two modeling levels not directly comparable, the adopted progress unit is well suited for measuring the concurrency of the rulesets. It can be seen that

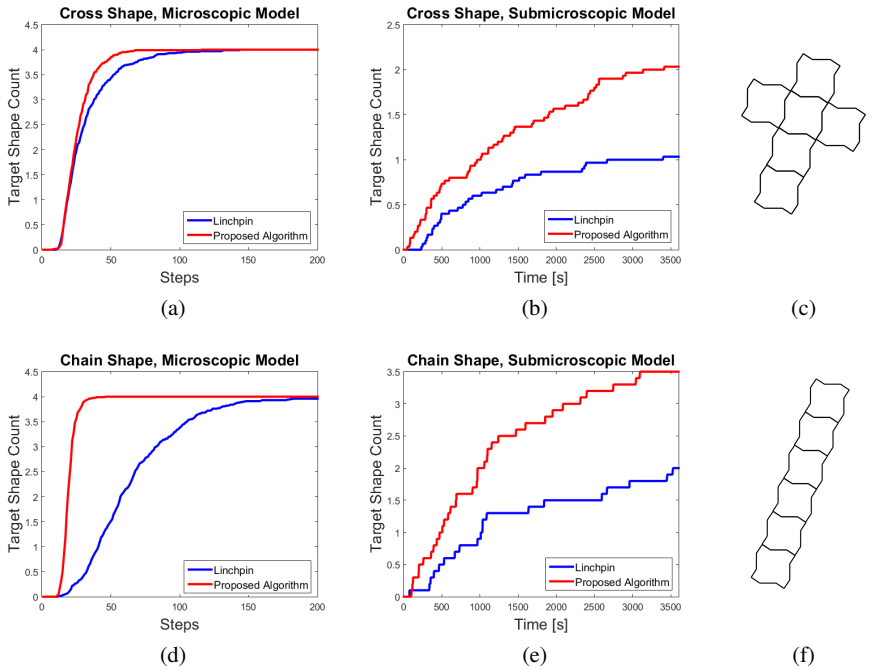


Fig. 3 – Results of microscopic ((a) and (d), 100 runs averaged) and submicroscopic models ((b) and (e), 30 runs averaged) for two target shapes, cross (c) and chain (f).

our proposed algorithm achieves higher assembly rates in all cases. Interestingly, the maximum yield of 4 is not obtained in the case of the submicroscopic simulations within the 1 hour simulated time. This can be ascribed to several reasons. First, it is observed in the submicroscopic simulation that larger structures with several corners trap other sub-assemblies and stall the SA process. In addition, as the structures grow the conditions diverge from perfect mixing since the shape of the sub-assemblies affects their orientation in the fluidic field and certain interactions tend to be less probable.

7 Conclusion

In this paper, we addressed the problem of synthesizing parallel rulesets for programmable SA of robotic modules. We employed an extended graph-grammar formalism to account for the morphology of the modules and proposed a formal synthesis algorithm to automatically synthesize rules. Using our new algorithm, we synthesized rulesets for two target structures. Studies on the synthesized rulesets in

simulation, using both non-spatial microscopic and spatial submicroscopic models, demonstrated the superior performance of our algorithm compared to the Linchpin algorithm [3], appropriately extended to be deployed on our robotic modules [7]. This evidenced the functionality of update rules in increasing the concurrency in the SA process resulting in higher assembly rates. In the future, we plan to conduct studies in simulation and on real robots to investigate the effects and mitigation strategies of propagation delays on the performance of the rulesets generated by our algorithm. Additionally, we plan to fully utilize our real experimental setup to conduct systematic real experiments involving up to 50 Lily robots.

Acknowledgements We gratefully acknowledge the contributions of Loic Waegeli and Brice Platerrier to the microscopic simulation framework. This work has been sponsored by the Swiss National Science Foundation under the grant numbers 200021_137838/1 and 200020_157191/1.

References

1. Cademartiri, L., Bishop, K.J.: Programmable self-assembly. Nature materials 14(1), 2–9 (2015)
2. Di Mario, E., Mermoud, G., Mastrangeli, M., Martinoli, A.: A trajectory-based calibration method for stochastic motion models. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4341–4347 (2011)
3. Fox, M., Shamma, J.: Probabilistic performance guarantees for distributed self-assembly. IEEE Transactions on Automatic Control 60(12), 3180–3194 (2015)
4. Gilpin, K., Rus, D.: Modular robot systems. IEEE Robotics & Automation Magazine 17(3), 38–55 (2010)
5. 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)
6. Haghighat, B., Martinoli, A.: Characterization and validation of a novel robotic system for fluid-mediated programmable stochastic self-assembly. To appear in IEEE/RSJ International Conference on Intelligent Robots and Systems (2016)
7. Haghighat, B., Platerrier, B., Waegeli, L., Martinoli, A.: Synthesizing rulesets for programmable robotic self-assembly: A case study using floating miniaturized robots. In: International Conference on Swarm Intelligence (ANTS). pp. 197–209 (2016)
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. Michel, O.: Webots: Professional mobile robot simulation. Advanced Robotic Systems 1(1), 39–42 (2004)
12. Rai, V., Van Rossum, A., Correll, N.: Self-assembly of modular robots from finite number of modules using graph grammars. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4783–4789. IEEE (2011)
13. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science 345(6198), 795–799 (2014)