Probabilistic performance guarantees for distributed self-assembly

Michael J. Fox Jeff S. Shamma

Abstract—In distributed self-assembly, a multitude of agents seek to form copies of a particular structure, modeled here as a labeled graph. In the model, agents encounter each other in spontaneous pairwise interactions and decide whether or not to form or sever edges based on their two labels and a fixed set of local interaction rules described by a graph grammar. The objective is to converge on a graph with a maximum number of copies of a given target graph. Our main result is the introduction of a simple algorithm that achieves an asymptotically maximum yield in a probabilistic sense. Notably, agents do not need to update their labels except when forming or severing edges. This contrasts with certain existing approaches that exploit information propagating rules, effectively addressing the decision problem at the level of subgraphs as opposed to individual vertices. We are able to obey more stringent locality requirements while also providing smaller rule sets. The results can be improved upon if certain requirements on the labels are relaxed. We discuss limits of performance in self-assembly in terms of rule set characteristics and achievable maximum yield.

I. INTRODUCTION

Self-assembly refers to the emergence of an ordered structure from the aggregate behavior of simpler constituent entities acting autonomously. It has been the subject of a great deal of research. The reasoning is twofold. First, understanding self-assembly generically may improve our understanding of natural self-assembling systems. Second, techniques applicable to the manufacture and operation of self-assembling engineered systems can potentially be developed. The hope is that the sort of scalability and reliability encountered in the natural examples can be realized in engineering applications. While interest in generic self-assembly dates back to at least the 1950's [1], the treatment of foundational possibility results has only begun to appear in the literature in recent years. What follows is intended as a brief overview of the state of the art and an attempt to frame the problems we address in a broader context.

We do not seek to address how self-assembling systems emerge in nature. We are concerned with the prospect of inducing self-assembly through careful programming on our part. Accordingly, our problem reduces to that of coordination. How can we enable the parts to achieve a global objective through

only local interactions? Obviously, the answer depends on what sort of local interactions are allowed. This issue is inherently application-specific. It is preferable for generic algorithms to be conservative in this regard. The ability to achieve self-assembly directives through local rules exclusively is a problem that is relevant to biology, robotics, manufacturing and other application areas. No particular model is especially canonical at present. The model introduced here will invariably be suited more to some application areas than others, e.g. it may not be meaningful to speak of programmability at all in the context of crystallization and polymerization processes that are governed solely by thermodynamics.

We propose algorithms for the synthesis of rules from a description of the assembly goal. Target assemblies are represented by specific graph topologies. A graph with at least as many agents as the target graph “assembles” the target graph by creating and severing edges according to preloaded local rules. If there are many more agents than the target requires then many copies of the target should be assembled. The intention is to study this stylized abstract decision problem in order to gain insight into the concrete problems that inspire it. In particular, emphasis is placed on issues arising from especially stringent locality constraints.

A key innovation in our approach is our allowing for probabilistic performance guarantees in the form of stochastic stability. These are the states in the support of the stationary distribution of a family of Markov chains as a perturbation term is taken to zero.

The system is modeled as a graph that evolves over time. Each vertex represents an anonymous agent. The finite number of vertices is fixed at the outset and the set of edges is dynamic. Each agent maintains an internal state taking on values from a finite set, which are represented as a labeling of the graph. At each time instant, two agents are selected at random. If there is a rule in the finite rule set that applies to the agents, they either apply the rule (changing the graph) or do nothing depending on the probability associated with that particular rule. If multiple rules apply, one is selected at random. The rules are described using the notation of graph grammars [2]. The objective is to realize a maximum number of disjoint maximal connected subgraphs isomorphic to a target graph. More plainly, a maximum yield of desirable assemblies is sought.

Constraints are expressed by restricting the types of rules allowed. The possibility of the parts utilizing their pairwise interactions to propagate edge information throughout their connected subgraph is excluded. If such consensus-type algorithms can be implemented reliably, then a centralized decision policy among subgraphs can be employed, circumventing the

Partially supported by AFOSR projects #FA95500810375, #FA95500910538, DARPA project #HR0011-10-1-0009, as well as the National Defense Science & Engineering Graduate Fellowship program.

Michael J. Fox was with the School of Electrical and Computer Engineering, michaelfox99@gmail.com

J.S. Shamma (corresponding author) is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, shamma@gatech.edu, www.prism.gatech.edu/~jshamma3, and with King Abdullah University of Science and Technology (KAUST), jeff.shamma@kaust.edu.sa.

localized coordination problem to some extent. Since there are contexts where this sort of information sharing may be infeasible, we address the possibility of self-assembly in the localized setting where agents do not propagate edge information. As some sort of coupled state evolution between the agents is necessary to achieve our objectives, we allow the two agents in a pairwise interaction to decide whether to form or sever an edge based on both of their labels. However, the pair may not update their labels unless they form or sever an edge. Thus, after two agents form a new edge, their existing neighbors cannot update their labels unless they simultaneously cease to be neighbors. This way, knowledge of the new edge cannot propagate. By excluding these rule types, we realize rule sets with small cardinality. Additionally, there are performance advantages in contexts where communication is costly, e.g. in terms of energy, or impossible altogether.

We also operate under reversibility constraints. Reversibility is a necessary property in many application areas. The constraint is tantamount to agents' not being able to construct permanent bonds. The impact of this constraint is explored by considering both processes that obey and neglect it.

Special attention is paid to the internal states that our algorithms require the agents to maintain. Internal states that can be recovered from the unlabeled graph (up to an isomorphism) are desirable because the internal state information may not be readily observable otherwise. The ability to infer internal states directly from graph topologies facilitates further analysis and control of the process. Another desirable feature of the internal states is uniqueness. Once an assembly is completed it is advantageous for each part to have a different label. If labels are not unique then the target graphs cannot include arbitrary labels. While we will only explicitly discuss unlabeled target graphs, it is understood that uniqueness of final labels is necessary and sufficient for the obvious extension to the case of labeled target graphs.

We suggest a simple procedure that gives maximum yields asymptotically as the total number of agents becomes large. A slightly more sophisticated procedure gives maximum yields for any number of parts, but introduces internal states that cannot be recovered from the graph. We suspect that algorithms giving both maximum yields and recoverable internal states exist. However, if we insist on uniqueness of states in complete assemblies as well, the situation is less clear. We show that a feature our analysis requires—the presence of a unique completing rule—can never be guaranteed for an algorithm with both unique and recoverable states.

The outline of this paper is as follows. In Section II we highlight other self-assembly research that is either prior or parallel to our own. Section III provides definitions of several basic concepts. We give a simple algorithm with asymptotically maximum yields in Section IV. We introduce an algorithm that always gives a maximum yield in Section V. We compare the two algorithms and comment on potential consequences for the internal states when we insist on maximum yields in Section VI.

This paper expands and develops on the work reported by the authors in [3], [4].

II. RELATED WORK

The synthesis problem for programmable self-assembly of graphs was introduced in [5]. There, the procedure depends upon communication between agents participating in an assembly and decisions are made according to a policy that relies on exhaustive search through all possible sub-assemblies. The notion of deadlock (multiple partial assemblies as undesirable equilibria) is also introduced. In [6] the formalism of graph grammars is first utilized in self-assembly of graphs and algorithms for synthesizing rules are presented. In particular, the *MakeTree* algorithm uses only constructive and destructive binary rules so that our communication constraint is observed. This procedure has a performance guarantee for all acyclic graphs when the number of agents is infinite. However, to avoid deadlock when the number of agents is finite, a disassociation rule must be added which depends upon implementation of a consensus-type algorithm. A primary contribution of our paper is the demonstration that these additional rules are not strictly necessary to achieve self-assembly. In [7], the problem of optimizing reaction rates to maximize the yields associated with the invariant state of a Markov chain is considered. Their optimization exerts control at the subassembly level rather than the agent level. They suggest additional “graph recognizer” rules as a means to actualize the approach via local rules using graph grammars based on algorithms from [6]. The approach suggested relies on solution of a bilinear programming problem that is NP-hard in the general case. Use of generic solvers giving local solutions of unknown quality is suggested. Furthermore, the problem size is exponential in the number of vertices in the target graph. In contrast, our algorithms generate rules in linear time and associated probabilities in constant time, which together are guaranteed to achieve or approximate the globally maximum yield. This stream of work has contributed many other results in this area including set-point regulation [8], and a robotic programmable parts testbed [9]. These works are similar in spirit to ours, as they emphasize restrictions on control over probabilistic encounters.

Designing self-assembly rules that are optimal with respect to convergence rates subject to a probabilistic performance constraint was considered in [10]. Stochastic stability also has been used as an equilibrium concept in a mildly related network formation game [11]. A similar notion of stability has been applied to the analysis of gene regulatory networks [12].

Another stream of research has modeled programmable self-assembly using cellular automata [13]. The generic algorithms are applicable to all assemblies that are filled, non-cantilevered, and convex in each layer. However, the agents are assumed to know their exact global position at all times. This can be guaranteed as long as the agents know their positions initially.

The tile, another form of cellular automata, is a model that has actually seen some experimental success [14]. Basic self-assembly and computation capabilities have been demonstrated with DNA-based tiles. This model also has various associated theoretical results relating to computational and self-assembly tasks; see for instance [15].

Numerous robotic self-assembling systems have been developed, notably [16] and [17]. Some mathematical formalization of these methods also has been done [18]. General global-to-local techniques for self-assembly are considered in [19]. A synopsis of various contributions in robotic self-assembly is available [20].

While most approaches to self-assembly have focused on structural assembly tasks, [21] has instead emphasized the function of resulting assemblies.

One of our algorithms, *Linchpin*, shares significant overlap with another algorithm developed simultaneously [22]. This fact helps to motivate our analysis of the potential conservatism of the approach both these algorithms take.

III. DEFINITIONS

A. Graph Grammars

This section succinctly reproduces the notion of graph grammars introduced in [6] with only slight differences.

A labeled graph is a triple $G = (V, E, \ell)$ where $V = \{1, \dots, N\}$ are vertices representing agents, $E \subset V \times V$ are pairs of vertices (or edges), and $\ell : V \rightarrow \mathcal{S}$ is a labeling function indicating the internal state of each agent, from a finite set $\mathcal{S}$. The number of agents is N . Agents are attached if their indices are among the pairs in E . A pair of vertices $\{x, y\} \in E$ is denoted by xy . The label $\ell(x)$ of an agent x is its internal state information from the finite set of states $\mathcal{S}$. We use the subscript notation V_G, E_G, ℓ_G to refer to respectively, the vertex set, edge set, and labeling function of a graph G . We also use $n_E(k)$ to refer to the neighbors of vertex k relative to the edge set E .

An unlabeled graph is simply the tuple $G = (V, E)$. The set of all unlabeled graphs with vertex set V is denoted $\mathcal{G}_V$.

Our self-assembly objectives will be related to E only. V will be static and ℓ will influence how E changes, but will not be material to our objectives intrinsically. In this framework, assemblies are network topologies. The precursor paper [3] used weighted graphs to confer geometric orientations on the edges, but we omit these details here in the interest of simplicity.

We say that an *isomorphism* exists between two graphs, or one graph is *isomorphic* to another when there exists a bijection $h : V_{G_1} \rightarrow V_{G_2}$ such that

$$ij \in E_{G_1} \Leftrightarrow h(i)h(j) \in E_{G_2}.$$

The function h is called a **witness**. The isomorphism is *label-preserving* if $\ell_{G_1}(x) = \ell_{G_2}(h(x)), \forall x \in V_{G_1}$.

Due to the vertices being identical atoms, two graphs that are isomorphic to each other represent the same assembly. Since we are concerned with self-assembly performance, our objective will be phrased in terms of isomorphism classes.

Given $I \subset V$ we define the *subgraph* $G \cap I = (V \cap I, E \cap I \times I, \ell|_{I \times I})$. We say that G contains H if a subgraph of G is isomorphic to H . A connected subgraph is maximal if there are no vertices in the original graph that could have been added to the subgraph while still leaving the subgraph connected. The term *assembly* is used as shorthand for a maximal connected subgraph.

Fig. 1. The rules in Example 3.1 can be applied successively to generate the cycle on the right. The subgraphs and witnesses can be inferred from the figure.

Definition 3.1: 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** and **right hand side** of r . The **size** of r is $|V_L| = |V_R|$.

Rules of size two are called *binary* rules. It is thought to be difficult in general to orchestrate interactions between many agents, so smaller rules are preferred. Here, only binary rules are considered throughout. If $E_L \subsetneq E_R$ a rule is called *constructive*. If $E_L \supsetneq E_R$ a rule is called *destructive*. Otherwise, the rule is *mixed*. Note that we define these set inequalities strictly, unlike some others. Visually a binary rule can be represented as

$$a - b \rightarrow c - d$$

where the letters are the labels and the vertices are suppressed. The left vertex of the left hand side corresponds to the left vertex of the right hand side, and similarly for the right vertices. A rule represents a local change in a graph, i.e. $|V_G| \geq |V_L|$. In the above example, labeled vertices a and b sever their edge and take new labels c and d , respectively.

Definition 3.2: 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 $h : I \rightarrow V_L$. The triple (r, I, h) is called an **action**.

Definition 3.3: When (r, I, h) is an action with $r = (L, R)$ on G , the **application** of (r, I, h) to G gives a new graph $G' = (V_G, E_{G'}, \ell_{G'})$ defined by

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

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

We write $G \xrightarrow{r, I, h} G'$ to indicate that G' was obtained from G via application of (r, I, h) .

Definition 3.4: The **complement** of a rule, $r = (L, R)$, is $\bar{r} = (R, L)$, so that $G \xrightarrow{r, I, h} G' \xrightarrow{\bar{r}, I, h} G'' = G$.

Given a set of rules Φ , the sequences of graphs obtained from successive application of the rules can be examined.

Example 3.1 (Simple cycle-building rules):

Consider the following set of constructive binary rules:

$$\Phi = \begin{cases} a - a \rightarrow b - c, & (r_1) \\ c - a \rightarrow d - e, & (r_2) \\ e - b \rightarrow f - g. & (r_3) \end{cases}$$

From the initial graph $G_0 = (\{1, 2, 3\}, \emptyset, \ell_0(\cdot) = a)$ there is only one possible trajectory obtainable by applying the unique applicable rule at each step, shown in Figure 1. $\square$

Example 3.2 (Binary communication):

Fig. 2. r_4 effectively acts as a communication step, updating the agent labeled d that the cycle has been closed.

Continuing with the previous example, consider the label d . When r_3 is applied, the chain closes into a cycle, but the vertex with label d is unaffected. Considering labels as representing the local information available to each agent, the agents with labels f and g know the exact structure of the graph since these labels are only adopted coinciding with r_3 . If we augment the rule set with a mixed rule:

$$\hat{\Phi} = \Phi \cup \{d - f \rightarrow h - f, \quad (r_4)\}$$

then the agent labeled d is apprised that the cycle is completed by its neighbor with label f , so that in the final graph, all agents are aware of the complete structure of the assembly they participate in. The effect of r_4 is illustrated in Figure 2. $\square$

The algorithms presented here synthesize a finite number of binary rules— each one being either constructive or destructive. They thus cannot exploit communication in the manner of the preceding example. This lack of communication protocols distinguishes from existing approaches. Those approaches rely on communication for avoidance of deadlocked states, which are discussed further below

If the number of vertices in the example were greater, it would be possible for r_3 to occur between two different sub-graphs, producing a long chain instead of a cycle. This reflects a fundamental limitation of finite binary rule sets [6]. For this reason, only acyclic assembly objectives are considered. It is also possible to realize something akin to communication using only constructive and deconstructive rules by forming a cycle with an agent that acts as an “observer” and propagates information. However, some additional assumptions would be necessary in order to avoid the basic problems presented by cycle-building rules.

Random pairwise selection dynamics induce an explicit Markov chain model via the application of binary rules.

B. Random pairwise selection dynamics

A random pairwise selection dynamic graph is a quadruple $\Sigma = (G_0, F, \Phi, \mathcal{R})$. The graph G_0 is an initial condition. Before defining F , first define the set

$$\text{PW}(G) = \{(x, y) : x, y \in V_G, x \neq y\},$$

i.e., pairs of distinct vertices. Then F is a mapping

$$F : \mathcal{G}_{V_{G_0}} \rightarrow \Delta[\text{PW}(G_0)],$$

where $\Delta[S]$ denotes the (simplex) set of full-support probability distributions over a finite set S . In other words, $F(G)$ maps unlabeled graphs to probabilities of pairwise vertex selections from V_G . When G is a labeled graph, we still write $F(G)$

Fig. 3. Successful realizations of $\{G_t\}$ occur with positive probability

Fig. 4. Unfortunately, the system in Example 3.3 can exhibit deadlock.

with the understanding that F depends only on the vertex set V_G and edge set E_G , but not on the labeling ℓ_G . The rule set is denoted by Φ . Finally, $\mathcal{R} : \Phi \rightarrow (0, 1]$ assigns a probability to each rule.

With these definitions, a random sequence of graphs $\{G_t\}_{t=1}^\infty$ is generated as follows:

- 1) Initialize with $t = 0$ and G_0 .
- 2) Increment t .
- 3) $F(G_t)$ is sampled, giving a pair of vertices $\{x, y\}$.
- 4) Let

$$\Phi_t = \{r \in \Phi : \exists h \text{ s.t. } (r, \{x, y\}, h) \text{ is an action on } G_{t-1}\}.$$

- 5) If $\Phi_t = \emptyset$ let $G_t = G_{t-1}$ and return to step 2.
- 6) Let $r \in \Phi_t$ be chosen at random, uniformly.
- 7) Let $G_{t-1} \xrightarrow{r, \{x, y\}, h} G'$.
- 8) Let

$$G_t = \begin{cases} G', & \text{with probability } \mathcal{R}(r) \\ G_{t-1}. & \text{with probability } 1 - \mathcal{R}(r) \end{cases}$$

- 9) Return to step 2.

We characterize the asymptotic behavior of $\{G_t\}$ for various choices of Φ and $\mathcal{R}$. The random sequence of selections, $F(G_t)$ is considered exogenous. Sampling from $F(G_t)$ gives an inherent stochasticity to the process even if $\mathcal{R}(\cdot) = 1$, i.e. no random behavior is introduced intentionally. Random pairwise selection dynamics can therefore be thought of as a model in which agents interact via random encounters and then behave according to the rules and their associated probabilities. The interaction probabilities depend on the current graph G_t . Since $F(G_t)$ is exogenous, there is only limited control over the trajectories of $\{G_t\}$. The long-run properties of the system can be influenced through Φ and $\mathcal{R}$. This model is appropriate for systems where agent motion is wholly or partly stochastic, such as in a liquid solution or on an air table.

C. The self-assembly problem

Let G_0 be an initial graph and $\hat{G}$ an unlabeled target graph with $|V_G| > |V_{\hat{G}}|$. Informally, the objective is to make as many disjoint copies of the target graph, $\hat{G}$, as possible. The *yield* of a graph G with respect to a target $\hat{G}$, $\mathcal{Y}_{\hat{G}}(G)$, is the

number of disjoint maximal connected subgraphs in G that are isomorphic to $\hat{G}$. Building on this definition, define the set:

$$\mathcal{G}_{V_{G_0}}^{\hat{G}} = \{G : V_G = V_{G_0}, \mathcal{Y}_{\hat{G}}(G) = \lfloor |V_{G_0}|/|V_{\hat{G}}| \rfloor\}$$

as the set of maximum yield graphs. For all the graphs in $\mathcal{G}_{V_{G_0}}^{\hat{G}}$, it is impossible for any rules to increase the number of completed assemblies. There is no preference for the remainder agents when $|V_{G_0}|$ is not an integer multiple of $|V_{\hat{G}}|$.

The self-assembly problem is, given F and G_0 , to find a set of rules Φ and associated probabilities $\mathcal{R}$ so that $\{G_t\}$ will enter and remain in $\mathcal{G}_{V_{G_0}}^{\hat{G}}$. There is no loss of generality in considering only unlabeled target graphs as opposed to labeled targets if the rules provide that each label in any complete assembly is unique.

Example 3.3 (Deadlock):

Consider the system $\Sigma = (G_0, F, \Phi, \mathcal{R})$ defined by

$$G_0 = (\{1, 2, 3, 4\}, \emptyset, \ell_0(\cdot) = a)$$

$$F \sim \text{uniform}$$

$$\Phi = \begin{cases} a & a \rightarrow b - c, & (r_1) \\ c & a \rightarrow d - e, & (r_2) \end{cases}$$

$$\mathcal{R}(\cdot) = 1.$$

Suppose $\hat{G} = (\{1, 2, 3\}, \{12, 23\})$. Figure 3 gives a possible trajectory for $\{G_t\}$. In this case, the process was successful since $G_t \in \mathcal{G}_V^{\hat{G}}$ for all $t \geq 2$. However, another possible trajectory is shown in Figure 4. In this case, the system has reached an undesirable stationary point and we have $G_t \notin \mathcal{G}_V^{\hat{G}}$ for all t . $\square$

Notice that in this example, each maximal connected subgraph of G_t is isomorphic to a *subgraph* of $\hat{G}$ — this is the phenomenon coined deadlock [5]. Deadlock is an issue because G_0 has finitely many vertices, so the supply of parts can become exhausted in undesirable graphs that are invariant under Σ . In [5], [6], [7] it is suggested that agents communicate amongst each other in order to determine whether their subgraph is complete. If it is not, then under certain conditions the agents are to engage in deconstruction to alleviate the deadlock. Presumably these communications could be expressed through binary graph grammars, although such constructions are not presented explicitly. The key insight from our algorithms is that deadlock can be avoided without requiring these communication protocols. Furthermore, the generic algorithm generating binary graph grammars for constructing trees in [6], *MakeTree*, generates the same number of constructive rules as our algorithms, namely $|E_{\hat{G}}|$. That algorithm is deadlock prone however, so more rules would be needed in order to realize performance guarantees when the number of parts is finite. Thus even when communication is not an obstacle, existing approaches that rely on it require strictly larger rule sets in the general case.

D. Reversibility

Depending on the constraints introduced on Φ and $\mathcal{R}$ it may not be possible to make $\mathcal{G}_V^{\hat{G}}$ an invariant set of the system Σ . In this case, only probabilistic statements about $\mathcal{Y}_{\hat{G}}(G_t)$ can be made.

One very natural constraint on Φ and $\mathcal{R}$ is related to the reversibility of the various rules. In many settings, reversibility is a necessary constraint in order for models to be realistic [23], [24]. The reversibility requirement ensures that no edge is permanent. In platforms where linkages are inherently volatile, the possibility of forming permanent edges cannot be assumed. Reversibility also guarantees that the induced Markov chain has a unique stationary distribution.

Definition 3.5: The pair $(\Phi, \mathcal{R})$ is **reversible** if $r \in \Phi$ implies that the complement $\bar{r} \in \Phi$.

This definition is analogous to reversibility in chemical reaction networks.

Although $\{G_t\}$ is a Markov process, the above definition of reversibility does not imply that $\{G_t\}$ is a reversible Markov process. A reversible Markov Process with state transition matrix P satisfies the detailed balance condition

$$P_{ij}\pi_i = P_{ji}\pi_j$$

for all i, j , where π_i and π_j are the stationary probabilities associated with states i and j , respectively. The notion of reversibility in Definition 3.5, in terms of the Markov process $\{G_t\}$ (with each possible graph a state), is $P_{ij} > 0 \Leftrightarrow P_{ji} > 0$.

Clearly it is impossible for $\{G_t\}$ to stay in $\mathcal{G}_V^{\hat{G}}$ when $(\Phi, \mathcal{R})$ is reversible. However, Φ and $\mathcal{R}$ can be chosen so that $\{G_t\}$ will be in $\mathcal{G}_V^{\hat{G}}$ with a high probability, or $\mathcal{Y}_{\hat{G}}(G_t)$ is close to $\lfloor |V|/|V_{\hat{G}}| \rfloor$ with high probability. These sorts of guarantees are expressed here using the concept of stochastic stability [25], [26]. The application of this particular notion of stochastic stability to self-assembly is novel, although a similar notion has been applied to the analysis of gene regulatory networks [12]. A review of stochastic stability and the resistance tree method is provided in the appendix.

E. Recoverable states

Before introducing our synthesis algorithms, we describe some desirable features of Φ and $\mathcal{R}$.

Of special interest are Φ and $\mathcal{R}$ that maintain a natural relationship between the unlabeled graph (V_{G_0}, E_{G_t}) and the labeling function at time t , ℓ_{G_t} . In particular, it is advantageous to be able to generate the labeled graph from the unlabeled graph so that the two agree up to a label-preserving isomorphism. In other words, we can infer the internal states (modulo symmetries) from graph topologies. There are some clear benefits to this feature. The full state of the system may be difficult to observe if internal states cannot be inferred from graph topologies, which would complicate any feedback control or diagnostics of the system.

Definition 3.6: Given G_0 and F , we say that Φ and $\mathcal{R}$ produce a Σ with **recoverable states** if there exists $\tilde{l} : 2^{V_G \times V_G} \times V_{G_0} \rightarrow \mathcal{S}$ such that for any G that is observed with positive probability under Σ there exists a label-preserving isomorphism between (V_{G_0}, E_G, ℓ_G) and $(V_{G_0}, E_G, \tilde{l}(E_G, \cdot))$.

Fig. 5. The labels in the fourth graph pictured are not unique.

In words, the function $\tilde{\ell}(E_G, \cdot)$ is able to reconstruct the labels of G up to an isomorphism given only the set of edges, E_G .

Particular emphasis is placed on the initial graph

$$G_0 = (\{1, 2, \dots, N\}, \emptyset, \ell_0(\cdot) = s_0).$$

Furthermore, since only F bounded away from zero are considered, Definition 3.6 will be a property of Φ and $\mathcal{R}$ alone. While the definition may appear a bit cumbersome, it will be straightforward to see if a particular Φ and $\mathcal{R}$ produce recoverable states only. As an illustration, it is easy to see that Example 3.1 produces recoverable states, whereas Example 3.2 does not. However, Example 3.2 also utilized a communicative mixed rule.

One algorithm considered will synthesize rules that introduce non-recoverable states due to the presence of multiple rules with the same left-hand side.

F. Uniqueness of final labels

A second feature of interest is uniqueness of labels in complete assemblies.

The self-assembly problem here does not require the complete assemblies to exhibit any particular labeling. The labels are merely auxiliary states. If labeled target graphs are sought then attention must be restricted to algorithms that provide unique final labels. If an algorithm leads to any redundancy among the labels in a complete assembly for some unlabeled target graph, then there is clearly some labeled target graph that the algorithm cannot realize. On the other hand, if the final labels are unique then arbitrary labeled targets are realizable. The definition below formalizes this observation.

Definition 3.7: Given G_0 , $\hat{G}$ and F , we say that Φ and $\mathcal{R}$ produce a Σ with **unique final states** if there exists an injective labeling function $\hat{\ell} : V_{\hat{G}} \rightarrow \mathcal{S}$ such that there is a label-preserving isomorphism between any subgraph observed with positive probability that is isomorphic to $\hat{G}$ and $\{V_{\hat{G}}, E_{\hat{G}}, \hat{\ell}\}$.

All of the examples encountered thus far produce unique final states. The example below illustrates a rule set that does not produce unique final labels.

Example 3.4 (Non-unique final states): Consider the system $\Sigma = (G_0, F, \Phi, \mathcal{R})$ defined by

$$G_0 = (\{1, 2, 3, 4\}, \emptyset, \ell_0(\cdot) = a)$$

$$F \sim \text{uniform}$$

$$\Phi = \begin{cases} a & a \rightarrow b - c, & (r_1) \\ c & c \rightarrow d - e, & (r_2) \end{cases}$$

$$\mathcal{R}(\cdot) = 1.$$

Suppose $\hat{G} = (\{1, 2, 3\}, \{12, 23, 34\})$; a chain of four vertices. Figure 5 illustrates a successful trajectory. Two agents have state b so the process does not give unique final labels. $\square$

IV. A SERIAL ALGORITHM

The first algorithm *Singleton*, provides self-assembly performance guarantees and satisfies both constraints (constructive/destructive binary, reversibility) and at the same time guarantees both recoverable states and unique final states.

The *Singleton* algorithm generates a rule set Φ from a target graph $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$. In [3] the authors presented a nearly identical algorithm as a standalone system without the notation of graph grammars. The algorithm is a recursion.

Algorithm 1 *Singleton*(V, E, k, s)

```

1:  $\Phi \leftarrow \emptyset$ 
2: if  $|n_E(k)| = 0$  then
3:   return  $(s, \Phi)$ 
4: else
5:    $\{v_j : j = 1, 2, \dots, |n_E(k)|\} \leftarrow n_E(k)$ 
6:    $\bar{s} \leftarrow s$ 
7:   for  $j = 1$  to  $|n_E(k)|$  do
8:      $\Phi \leftarrow \Phi \cup \{\bar{s} \quad 0 \rightleftharpoons (s+1) - (s+2)\}$ 
9:      $\bar{s} \leftarrow s+1$ 
10:     $s \leftarrow s+2$ 
11:    let  $(V^j, E^j)$  be the component of  $(V, E - \{kv_j\})$ 
        containing  $v_j$ 
12:     $(s_j, \Phi_j) \leftarrow \text{Singleton}(V^j, E^j, v_j, s)$ 
13:     $\Phi \leftarrow \Phi \cup \Phi_j$ 
14:     $s \leftarrow s_j$ 
15:  end for
16: end if
17: return  $(s, \Phi)$ 

```

In line 8, the reversible arrows ($\rightleftharpoons$) indicate that the rule is to be understood as two complementary rules. Evident from line 8 is the inspiration for the name: all constructive rules involve a vertex with label 0—the label reserved for vertices not participating in any edges.

Running *Singleton*($V_{\hat{G}}, E_{\hat{G}}, k, 0$) for any $k \in V_{\hat{G}}$ gives a rule set. The variable s maintains the largest label assigned by the recursive application of the algorithm, and so $s = 0$ in the initial call.

For a target graph $\hat{G}$ that is connected and acyclic, the algorithm proceeds as follows. The vertex k is the root of the tree. The algorithm iterates through k 's neighbors one at a time. Assuming $|V_{\hat{G}}| \geq 2$, the first rule is always

$$0 \quad 0 \rightleftharpoons 1 - 2,$$

where 1 is the label assigned to the vertex that will play the role of the root and 2 is the label of its neighbor. If this neighbor has no other edges, the algorithm proceeds to the next neighbor and adds the rule

$$1 \quad 0 \rightleftharpoons 3 - 4,$$

so that the vertex playing the role of k forms an edge with a singleton thereby filling a vacancy, and updates its label. The algorithm continues in this manner for each neighbor of

Fig. 6. The system in Example 4.1 can assemble itself through application of the odd-numbered (constructive rules) in order, but may then apply destructive rules and disassemble.

the k -vertex. If one of the neighbors has neighbors other than k then a recursive call to *Singleton* is made, treating the neighbor as the k -vertex (i.e., the root) of the graph obtained by making a cut between the original k -vertex and the neighbor. The largest label s is kept track of so that each new vertex added is assigned a unique label.

At a high level, the algorithm succeeds because each singleton added on is able to determine its role in the target graph from the label of the vertex it forms an edge with. This information determines what vacancies, if any, it has for additional edges. The internal states thus provide only limited information about the overall structure of the subgraph that agents participate in. After a singleton forms an edge and receives a role, it may update its state as it fills vacancies, but will not know whether its neighbors have filled their vacancies.

The rule set returned by *Singleton* is not necessarily immune to deadlock, but an appropriate choice of $\mathcal{R}$, the function assigning rule probabilities, will be accompanied by a strong performance guarantee.

Below is a simple example of the rule set constructed by the *Singleton* algorithm.¹

Example 4.1 (*Singleton* algorithm): Consider the target graph $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ defined by

$$V_{\hat{G}} = \{1, 2, 3, 4, 5, 6\}$$

$$E_{\hat{G}} = \{12, 13, 14, 15, 56\}.$$

Let Φ be the rule set returned by *Singleton*($V_{\hat{G}}, E_{\hat{G}}, 1, 0$)

$$\Phi = \begin{cases} 0 & 0 \Rightarrow 1-2, & (r_1, r_2) \\ 1 & 0 \Rightarrow 3-4, & (r_3, r_4) \\ 3 & 0 \Rightarrow 5-6, & (r_5, r_6) \\ 5 & 0 \Rightarrow 7-8, & (r_7, r_8) \\ 8 & 0 \Rightarrow 9-10. & (r_9, r_{10}) \end{cases}$$

Now consider a complete Σ as follows

$$G_0 = (\{1, 2, 3, 4, 5, 6\}, \emptyset, \ell_0(\cdot) = 0)$$

$$F \sim \text{uniform}$$

$$\mathcal{R}(\cdot) = 1.$$

Figure 6 illustrates an execution of this system that successfully assembles via the application of the constructive rules in order. Unfortunately, in the last graph the application of a destructive rule has taken the system out of $\mathcal{G}_V^{\hat{G}}$. $\square$

¹See the appendix for a walkthrough of how the rule set is constructed.

A. Analysis of *Singleton*

A consequence of Φ being a reversible set of rules is that completed assemblies cannot be made stable—removing a part from a complete assembly and lowering the yield by one occurs with positive probability. This phenomenon was observed in the preceding example. However, reversibility has the benefit of freeing the system from deadlock. Properly balancing these two attributes via $\mathcal{R}$ will be necessary in order to provide any sort of performance guarantee for *Singleton* systems.

Consider an arbitrary connected, acyclic $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ and the initial graph

$$G_0 = (\{1, 2, \dots, N\}, \emptyset, \ell_0(\cdot) = 0)$$

with each $r \in \Phi$ having probability

$$\mathcal{R}(r) = \begin{cases} a_r, & r \text{ is constructive} \\ \epsilon, & r \text{ is destructive} \end{cases}$$

where Φ is obtained from *Singleton*($V_{\hat{G}}, E_{\hat{G}}, k, 0$) for any $k \in V_{\hat{G}}$. The values $a_r \in (0, 1]$ are arbitrary constants.

Destructive rules are executed with small probability ϵ . Our analysis characterizes the set of *stochastically stable* states, i.e., states with non-vanishing probability as $\epsilon \downarrow 0$ (see Appendix).

The random pairwise selection dynamics with the above rule probabilities is a regular perturbed Markov process over the space of graphs, henceforth referred to as P^ϵ . In particular, each state of P^ϵ is an isomorphism class of graphs. The unperturbed process, P^0 , is obtained by removing the destructive rules from Φ and $\mathcal{R}$. The following result is immediate.

Lemma 4.1: The absorbing states of P^0 are all states where each subgraph of G is isomorphic to a subgraph of $\hat{G}$. Either $|n_{E_G}(i)| \geq 1, \forall i \in V_G$, or there exists exactly one $i \in V_G$ such that $|n_{E_G}(i)| = 0$.

In other words, every assembly in every absorbing state of P^0 is a partial or complete assembly. The only circumstance where an agent without any edges persists is when all other agents participate in complete assemblies. Otherwise, the singleton vertex and some other vertex would comprise a left hand side of a constructive rule in Φ , which would contradict the state's being absorbing. Clearly P^0 has many states in $\mathcal{G}_V^{\hat{G}}$, many having nearly maximum yields, but also has quite a few deadlock states with low yields. Only the absorbing states of P^0 can potentially be stochastically stable states of P^ϵ .

The performance guarantees for the *Singleton* algorithm rely on the following theorem:

Theorem 4.1: The stochastically stable states of P^ϵ are the absorbing states of P^0 with the minimum number of disjoint maximal connected subgraphs. In particular, there are $\lfloor |V_{G_0}| / |V_{\hat{G}}| \rfloor$ such subgraphs in each and every one of the stochastically stable states.

Proof: The proof is based on the construction of rooted trees for the claimed class of stochastically stable states and comparison with the trees corresponding to all other absorbing states. Note that the trees referred to in this proof are directed spanning trees with the vertices being the absorbing states of P^0 , per the resistance tree method (see Appendix). They

Fig. 7. The states $z_m, m \in \{3, 4, 5\}$, $N = 10$, $|V_{\hat{G}}| = 4$.

should not be confused with the forests of trees that comprise the absorbing states themselves.

The resistance between an ordered pair of absorbing states of P^0 is the minimum number of state transitions with probability proportional to ϵ among all directed paths from the first state to the second. The stochastic potential of an absorbing state is the minimum sum of resistances among the edge sets of directed spanning trees rooted at that state. The states with minimum stochastic potential are the stochastically stable states.

If $|V_{\hat{G}}| = 2$ then all absorbing states have maximum yield and the proof is immediate, so assume $|V_{\hat{G}}| \geq 3$. Let Z_0 be the absorbing states of P^0 . Partition Z_0 into disjoint sets Z_m where each state in Z_m has m assemblies, so that

$$\bigcup_{m \in \mathcal{M}} Z_m = Z_0,$$

where

$$\mathcal{M} = \{\lceil |V_{G_0}|/|V_{\hat{G}}| \rceil, \lceil |V_{G_0}|/|V_{\hat{G}}| \rceil + 1, \dots, \lceil |V_{G_0}|/2 \rceil\}.$$

The rooted trees for each absorbing state contain $|Z_0| - 1$ edges. There is nonzero resistance associated with each of these edges because the states are all absorbing. For P^ϵ , the resistance is at least one for each edge. A rooted tree satisfying this minimum resistance of $|Z_0| - 1$ can be constructed for each state in $Z_{\lceil |V_{G_0}|/|V_{\hat{G}}| \rceil}$, the set of absorbing states associated with the minimum number of disjoint maximal connected subgraphs.

To shorten the proof attention is restricted to the case where both $N = |V_{G_0}|$ and $|V_{\hat{G}}|$ are even. A similar construction exists for the cases where either N , $|V_{\hat{G}}|$, or both are odd. Let $z_{N/2} \in Z_{N/2}$ be the state with all assemblies as pairs of vertices. Let $z_{N/2-1} \in Z_{N/2-1}$ be the state arrived at by applying the appropriate destructive rule on one of the pairs and then applying constructive actions to attach the two free vertices to one of the other pairs. Proceed like this for each $z_m \in Z_m$, arriving at z_{m-1} by breaking up a pair and transferring the pieces to the largest possible assembly. This requires a destructive rule followed by two constructive rules. Figure 7 illustrates an example of this procedure.

First, construct the tree rooted at $z_{\lceil N/|V_{\hat{G}}| \rceil}$ as follows. There are directed edges corresponding to the z_m as follows:

$$z_{\lfloor N/2 \rfloor} \rightarrow z_{\lfloor N/2 \rfloor - 1} \rightarrow \dots \rightarrow z_{\lceil N/|V_{\hat{G}}| \rceil}.$$

Each edge $z_m \rightarrow z_{m-1}$ represents breaking up one assembly of two agents (resistance of one), and then having those vertices form together.

Thus far, the tree includes only a single representative state z_m for each set Z_m . What remains to be shown is that all remaining states in each Z_m can reach z_m via a path through states in Z_m with all edges having resistance 1. Consider a state $y \in Z_m, y \neq z_m$. Each z_m consists of only two-vertex assemblies, completed assemblies, and at most one other assembly. Let x_m refer to the largest incomplete assembly of z_m (let it be a two-vertex assembly if that is the largest). The path to z_m for each of the two cases for y are as follows.

Case 1: There exists a maximal connected subgraph of y that is isomorphic to x_m . In this case, take the smallest assembly with more than two vertices and shift a vertex to the largest incomplete assembly other than the one that is isomorphic to x_m . We continue this process until we obtain z_m . Each step in the process involves one destructive rule and therefore an edge with resistance one linking to a distinct absorbing state in Z_m .

Case 2: There is no maximal connected subgraph of y that is isomorphic to x_m . In this case, take the smallest assembly with more than two vertices and shift a vertex to the largest maximal connected subgraph that is isomorphic to a subgraph of x_m . Proceeding like this, a maximal connected subgraph that is isomorphic to x_m is obtained, which leads to the first case.

This process can be repeated for all of the states in Z_m , avoiding any redundancies so that precisely $|Z_m| - 1$ edges are formed. Applying this technique for all m gives a rooted tree with each edge having resistance one so that $z_{\lceil N/|V_{\hat{G}}| \rceil}$ is stochastically stable.

This construction can be extended to all the other states in $Z_{\lceil N/|V_{\hat{G}}| \rceil}$. For an arbitrary state $z' \in Z_{\lceil N/|V_{\hat{G}}| \rceil}, z' \neq z_{\lceil N/|V_{\hat{G}}| \rceil}$, construct the tree just as above for the states in the sets $Z_m, m > \lceil N/|V_{\hat{G}}| \rceil$ and for $z_{\lceil N/|V_{\hat{G}}| \rceil}$. Then, insert the edges between $z_{\lceil N/|V_{\hat{G}}| \rceil}$ and z' in the reverse direction. Finally, apply the exact same procedure as above for the remaining states in $Z_{\lceil N/|V_{\hat{G}}| \rceil}$, again avoiding any redundancies. These trees will also all have resistance one at every edge so that all of $Z_{\lceil N/|V_{\hat{G}}| \rceil}$ is stochastically stable.

For any state in $Z_m, m \neq \lceil N/|V_{\hat{G}}| \rceil$ the rooted trees all must include an edge that goes from a state with a smaller number of assemblies to a state with a larger number of assemblies. This can only be accomplished by application of two consecutive destructive rules corresponding to an edge with resistance two. Since all other edges have resistance of at least one, all of these rooted trees have resistance equal to at least $|Z_0|$. Therefore, the stochastically stable states are precisely $Z_{\lceil N/|V_{\hat{G}}| \rceil}$, the states with the minimum number of assemblies. ■

In general, not all stochastically stable states of P^ϵ are in $\mathcal{G}_V^{\hat{G}}$, but there is the following special case.

Corollary 4.1: If $N = m|V_{\hat{G}}|$ for some $m \in \mathbb{Z}_+$ then all stochastically stable state have all maximal connected subgraphs of G isomorphic to $\hat{G}$.

This result is immediate from the preceding Theorem. Of course, such a strong result will not hold when N is not an integer multiple of $|V_{\hat{G}}|$. This leads to a curiosity, namely that the minimum yield among the stochastically stable states can be decreased by increasing N . This is somewhat surprising given that *Singleton* generates rules without any consideration of N , and increasing N makes more parts available for assembly. Thus, performance can be poor when N is both not much larger than $|V_{\hat{G}}|$ and not an integer multiple of $|V_{\hat{G}}|$. Nevertheless, the situation is much better when N is large.

Theorem 4.2: All stochastically stable states of P^ϵ have no more than $(|V_{\hat{G}}| - 1)^2$ vertices not part of a connected subgraph isomorphic to $\hat{G}$. Further, at most $|V_{\hat{G}}| - 1$ subassemblies are incomplete.

Proof: Let $N = (|V_{\hat{G}}| - 1)^2$. The maximum number of incomplete assemblies among stochastically stable states is $|V_{\hat{G}}| - 1$, with $|V_{\hat{G}}| - 1$ vertices in each assembly. Any configuration with more incomplete assemblies would have more total assemblies, which is not stochastically stable by Theorem 4.1. Each increase of N by one, must add one complete assembly and reduce the number of vertices not participating in complete assemblies by $|V_{\hat{G}}| - 1$. This continues until we reach $N = |V_{\hat{G}}|(|V_{\hat{G}}| - 1)$ and all vertices participate in complete assemblies in the stochastically stable states. This process repeats for $|V_{\hat{G}}|(|V_{\hat{G}}| - 1) + 1$ through $|V_{\hat{G}}|^2$ so that it is easy to show by induction that $(|V_{\hat{G}}| - 1)^2$ is always the maximum number of vertices not part of complete assemblies. ■

B. Remarks

Theorem 4.2 upper bounds the number of reject assemblies for all N . When $N \gg |V_{\hat{G}}|$ the yield of *Singleton* is only negligibly different from the maximum. It is an open question as to whether or not this guarantee can be improved upon without compromising on the constraints or features of the internal states. Empirically, we have found that introducing resistances greater than one for destructive rules applied further from the k -vertex can improve performance in some simulations, but these results are yet to be formalized.

The basic action of the *Singleton* process is to place more probability weight on assembly than disassembly. The system tends toward assembly because of this. At the same time, the positive probabilities associated with disassembly alleviate deadlock. The shortcomings of the *Singleton* process are related to the fact that it views each edge the same way. This is why when

$$|V_{\hat{G}}| < N < 2|V_{\hat{G}}|$$

the probability of observing two incomplete assemblies is comparable to the probability of observing one complete assembly and one incomplete assembly. Both situations exhibit the same number of edges and it is the number of edges that the process drives down, or equivalently, the total number of assemblies.

The perturbations can represent either the deliberate actions of the agents as part of a stochastic control policy, or the effects of exogenous process noise. Varying the parameter ϵ achieves a tradeoff between running time and quality in the self-assembly process. As ϵ shrinks, the process spends an increasing portion of time in the stochastically stable states, which we have shown correspond to near-maximum yields. On the other hand, it will take longer to “back out” out of states corresponding to poor yields, since such transitions are limited by the rate at which ϵ -probability deconstructive rules are applied.

The next section describes an algorithm for synthesizing Φ when non-recoverable states are allowed. This process is able to improve upon the performance of the *Singleton* process by treating some edges differently than others.

V. A PARALLEL ALGORITHM

Under random pairwise comparison dynamics the sequence of graphs $\{G_t\}$ is random. The strongest possible performance guarantee that can be provided is that $\{G_t\}$ converges to the set $\mathcal{G}_V^{\hat{G}}$ almost surely. If Φ must be reversible, then this sort of guarantee is not possible. In the previous section an alternative for this circumstance in the form of stochastic stability was described. Stochastic stability provides a continuum of systems such that the stationary probability of observing a mostly complete $\{G_t\}$ goes to one as the parameter ϵ goes to zero. This section aims to improve upon the result of the previous. In particular, an algorithm is suggested that only admits stochastically stable states in $\mathcal{G}_V^{\hat{G}}$ —the maximum yield states. The algorithm gives a reversible set of binary constructive and deconstructive rules rule set, but will in most cases introduce non-recoverable states. The process will always have unique final states. If a single irreversible rule is allowed, then the system will converge to $\mathcal{G}_V^{\hat{G}}$ almost surely.

Non-recoverable states are often associated with non-uniqueness of the left hand sides in Φ . The *Linchpin* algorithm will generate non-recoverable states precisely because of this issue. Like *Singleton*, *Linchpin* is a recursion that generates Φ from a target graph $\hat{G}$ and an initial vertex k .

To obtain a rule set, run *Linchpin*($V_{\hat{G}}, E_{\hat{G}}, k, 0$) for any $k \in V_{\hat{G}}$. The target graph $\hat{G}$ must be connected and acyclic.² The defining feature of rule sets generated from *Linchpin* is the presence of a completing rule. That is, every assembly is completed by application of the same rule. The next example illustrates this feature and compares the rule sets generated by *Linchpin* and *Singleton*.

Example 5.1 (Completing rules): Suppose $\hat{G} = (\{1, 2, 3, 4\}, \{12, 23, 34\})$; a chain of four vertices. Furthermore let

$$\begin{aligned} \Phi_S &= \text{Singleton}(V_{\hat{G}}, E_{\hat{G}}, 2, 0) \\ &= \begin{cases} 0 & 0 \Rightarrow 1 - 2, & (r_1, r_2) \\ 2 & 0 \Rightarrow 3 - 4, & (r_3, r_4) \\ 1 & 0 \Rightarrow 5 - 6, & (r_5, r_6) \end{cases} \end{aligned}$$

²See the appendix for a walkthrough of how the rule set is constructed.

Algorithm 2 $\text{Linchpin}(V, E, k, s)$

```

1:  $\{v_j : j = 1, 2, \dots, |n_E(k)|\} \leftarrow n_E(k)$ 
2: for  $j = 1$  to  $|n_E(k)|$  do
3:   if  $|n_E(v_j)| \geq 2$  then
4:     let  $(V^j, E^j)$  be the component of  $(V, E - \{kv_j\})$ 
       containing  $v_j$ 
5:      $(s_j, \Phi_j) \leftarrow \text{Linchpin}(V^j, E^j, v_j, s)$ 
6:      $s \leftarrow s_j$ 
7:   else
8:      $s_j \leftarrow 0$ 
9:      $\Phi_j \leftarrow \emptyset$ 
10:  end if
11: end for
12:  $\Phi \leftarrow \Phi_1 \cup \{s_1 \mid 0 \Rightarrow (s+1) - (s+2)\}$ 
13:  $s \leftarrow s+2$ 
14: for  $j = 2$  to  $|n_E(k)|$  do
15:    $\Phi \leftarrow \Phi \cup \Phi_j \cup \{s_j \mid s \Rightarrow (s+1) - (s+2)\}$ 
16:    $s \leftarrow s+2$ 
17: end for
18: return  $(s, \Phi)$ 

```

Fig. 8. Assembly of $\hat{G}$ via Φ_S with r_5 being the last rule applied. Clearly the order of r_3 and r_5 can be reversed.

and

$$\begin{aligned}
\Phi_L &= \text{Linchpin}(V_{\hat{G}}, E_{\hat{G}}, 2, 0) \\
&= \begin{cases} 0 & 0 \Rightarrow 1 - 2, & (r_1, r_2) \\ 0 & 0 \Rightarrow 3 - 4, & (r_3, r_4) \\ 2 & 4 \Rightarrow 5 - 6, & (r_5, r_6) \end{cases}
\end{aligned}$$

Figure 8 shows one trajectory for Φ_S . In this case, the process culminates in application of r_5 . However, the order of r_5 and r_3 can also be reversed. The consequence is that there is no unique completing rule. For this example, the starting vertex argument k could have been chosen differently in the `Singleton` algorithm and generated a rule set with a unique completing rule, but it is easy to construct $\hat{G}$ for which no such choice exists.

Rule sets generated by the `Linchpin` algorithm always give self-assembly trajectories that culminate in a unique completing rule. This is true irrespective of the starting vertex k .

Fig. 9. Assembly of $\hat{G}$ via Φ_L always culminates with r_5 .

Figure 9 illustrates this phenomenon for our present example. It is this feature of the `Linchpin` algorithm that will enable an improvement upon the guarantees for the `Singleton` algorithm. $\square$

The `Singleton` algorithm does not have a unique completing rule because self-assembly proceeds outward from the starting vertex k . Since the target graph likely has many branches, any of a number of leaves can be added on last. In contrast, `Linchpin` assembles from each leaf in towards the k -vertex so that the overall process culminates with two sub-graphs joining together. These subgraphs are themselves assembled recursively in the same manner.

Recall that the principal action of `Singleton` is to seek absorbing states with a minimum number of assemblies. This process allows up to $|V_{\hat{G}}| - 1$ incomplete assemblies. When N is not large this can be a significant limitation. The `Linchpin` algorithm can easily circumvent this limitation due to the presence of a unique completing rule. The suppression of the complement of this completing rule will be the key to achieving this end.

Let $\hat{s}$ be the label returned by `Linchpin`. Then there is one rule whose left hand side contains this label—the complement of the completing rule, $\hat{r}$. Consider the following rule probabilities

$$\mathcal{R}(r) = \begin{cases} a_r, & r \neq \hat{r} \\ \epsilon, & r = \hat{r} \end{cases}$$

where $a_r \in (0, 1]$ are arbitrary constants. As with the `Singleton` algorithm, this choice of $\mathcal{R}$ gives a regular perturbed Markov process, P^ϵ . The unperturbed process P^0 is obtained by removing $\hat{r}$ from Φ .

Before we analyze the random process induced by Φ for this choice of $\mathcal{R}$, we establish some properties of Φ . First, we show that the rule set returned by `Linchpin` is, in principle, capable of constructing $\hat{G}$.

Lemma 5.1: For any tree $\hat{G}$, let Φ be given by `Linchpin` ($V_{\hat{G}}, E_{\hat{G}}, k, 0$). Then there exists a sequence of constructive actions in Φ that, applied to $G_0 = \{V_{\hat{G}}, \emptyset, \ell_{G_0}(\cdot) = 0\}$, result in $\hat{G}$.

Proof: The proof is by induction on the depth of the tree rooted at k . If k has no neighbors then the algorithm returns no rules and the lemma is satisfied vacuously. The base case is a depth of 1. In this case, $\hat{G}$ is a star with k at its center. Line 1 assigns any order to the neighbors of k . Lines 2-11 iterate through these neighbors and in this case always execute lines 8 and 9 that assign $s_j = 0$ and $\Phi_j = \emptyset$ for each neighbor v_j because, by assumption, each v_j has no neighbors other than k . Line 12 gives the first rules $\{0 \mid 0 \Rightarrow 1 - 2\}$. The part assigned state 2 will continue on with the role of k . Each rule added in lines 14-17 adds another singleton to k . The lemma is satisfied by applying the constructive rules from Φ in the order that they were added.

For the induction step assume that `Linchpin` satisfies the lemma (by applying the constructive rules in the order they were added to Φ) when the depth of the tree rooted at k is at most D . Next, suppose that the depth of the tree rooted at k is $D + 1$. Some of k 's neighbors may have neighbors other

than k . In this case, the recursive call to `Linchpin` on line 5 is made with v_j as the new k -vertex. Since the depth of the subtree rooted at v_j obtained in line 4 is at most D , we get a sequence of rules that build this subtree by assumption. Note that s_j is the state of v_j in the completed subtree. Also note that each recursive call introduces only unused labels. Line 12 gives a rule that adds k as a singleton to the completed subtree for v_1 . Lines 14-17 now add the remaining subtrees to k . If a subtree is just v_j then $s_j = 0$ and a singleton is added to k . If the subtree for v_j is not a singleton then s_j is the state of v_j once that subtree has finished assembling. The lemma is once again satisfied by applying the constructive rules from Φ in the order that they were added, completing the proof. ■

The convergence proof depends on one additional property of `Linchpin`— the presence of a unique *completing rule*. Since the rule set is reversible, this property is equivalent to the disassembly of any complete assembly beginning with a unique rule.

Lemma 5.2: For any tree $\hat{G}$, let Φ be given by `Linchpin` ($V_{\hat{G}}, E_{\hat{G}}, k, 0$). Let G be the labeled graph obtained by applying the constructive rules in Φ (in the order they were added) to $G_0 = \{V_{\hat{G}}, \emptyset, \ell_{G_0}(\cdot) = 0\}$. Then there is exactly one rule in Φ applicable to G , it is deconstructive, and it involves the vertex k . Furthermore none of the labels in G appear in the left hand sides of the constructive rules in Φ .

Proof: The proof is again by induction on the depth of the tree rooted at k . For the base case, examine lines 12-17. Each rule adds a singleton to k and increases k 's state by 2. Since k 's state has changed, only the last deconstructive rule added still applies. The final label for k is also greater than any of the labels on the left hand sides of the constructive rules. For the inductive step assume the lemma holds for trees rooted at k with depth no greater than D and suppose that the tree rooted at k has depth $D+1$. Then lines 2-11 give rules that produce subtrees satisfying the lemma. Now, just as in the base case, when these subtrees are added to k , v_j 's label is increased so that it no longer has any applicable rules unless it was the most recent addition. The final label for k is also again greater than any of the left hand sides of constructive rules. Only one rule applies to the finished product and it is the deconstructive rule severing k and $v_{|n_E(k)|}$ — the `Linchpin`. ■

The requirement on the left hand sides of constructive rules is to ensure that the assembly will not “overassemble” in the presence of additional parts. The above lemma assumed that the vertex set of the graph matched the vertex set of target graph and that the rules were applied in a specific order. The next lemma extends the unique completing rule property to general graphs where rules are not necessarily applied in order.

Lemma 5.3: For any tree $\hat{G}$, let Φ be given by `Linchpin` ($V_{\hat{G}}, E_{\hat{G}}, k, 0$). Let G be any complete assembly obtained by applying constructive rules in Φ to $G_0 = \{V_{\hat{G}}, \emptyset, \ell_{G_0}(\cdot) = 0\}$. Then G is a label-preserving isomorphism of the graph obtained in Lemma 5.2.

Proof: The proof is yet again by induction on the depth of the tree rooted at k . The base case of a star with center k gives constructive rules that must be applied in the exact order they were added. This implies that the final labels are unique within subassemblies. For the inductive step assume the labels

are unique for depth D . When the depth is $D+1$ the recursive calls give, by assumption, subtrees with unique labels. Then, similar to the base case, the subtrees must be combined in a specific order so that the final labels are again unique. ■

Note that the different subtrees can be completed in any order so that `Linchpin` gives rules which allow for parallel self-assembly. The rules do not need to be applied in exactly the order they were added to Φ (as in Lemma 5.2). With these three lemmas in hand we proceed toward our main result, the performance guarantees for random pairwise selection using rule sets generated by `Linchpin`.

While the unperturbed process in the case of `Singleton` was especially deadlock-prone, this is not true of the unperturbed `Linchpin` process. In fact, the stationary distribution of P^0 places positive probability on states in $\mathcal{G}_V^{\hat{G}}$ only. Of course, P^0 is not reversible. The performance guarantee for P^0 are established first, since the analysis for P^ϵ will be a straightforward extension of these.

Consider an arbitrary connected, acyclic $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ and the initial graph

$$G_0 = (\{1, 2, \dots, N\}, \emptyset, \ell_0(\cdot) = 0).$$

The rule set Φ and rule probabilities $\mathcal{R}$ are as specified above, which gives the unperturbed process $\{G_t\}$. Recall that $\mathcal{Y}_{\hat{G}}(G_t)$ is the yield of $\hat{G}$ for the process at time t .

Lemma 5.4: For the unperturbed `Linchpin` process, $\mathcal{Y}_{\hat{G}}(G_t)$ is nondecreasing in t .

Proof: Suppose for the sake of contradiction that there exists $\tau > 0$ such that $\mathcal{Y}_{\hat{G}}(G_\tau) < \mathcal{Y}_{\hat{G}}(G_{\tau-1})$. The only way that the number of maximal connected subgraphs of $G_{\tau-1}$ that are isomorphic to $\hat{G}$ can decrease is if $\hat{r}$ is applied, but $\hat{r} \notin \Phi$ for the unperturbed process. ■

Next we establish that $\mathcal{Y}_{\hat{G}}(G_t)$ increases with positive probability.

Lemma 5.5: Suppose that $\mathcal{Y}_{\hat{G}}(G_t) < \lfloor N/|V_{\hat{G}}| \rfloor$, then there exists a length of time T and a probability $p > 0$ such that $\Pr[\mathcal{Y}_{\hat{G}}(G_{t+T}) > \mathcal{Y}_{\hat{G}}(G_t)] = p$.

Proof: Since only $p > 0$ is required, only one trajectory with positive probability must be found. If there are $|V_{\hat{G}}|$ vertices with label 0 then appropriate vertices can be selected so that constructive rules are applied. If there are insufficient agents then destructive rules can be applied to incomplete assemblies to free up parts. In either case, the associated probability is positive and T is simply the number of rules applied. ■

The following result is now immediate.

Theorem 5.1: For the unperturbed `Linchpin` process, $G_t \rightarrow \mathcal{G}_V^{\hat{G}}$ almost surely.

A subset of $\mathcal{G}_V^{\hat{G}}$ is the only recurrent class of the process. It follows that these are precisely the stochastically stable states of the perturbed process.

Theorem 5.2: The stochastically stable states of the perturbed `Linchpin` process are a subset of $\mathcal{G}_V^{\hat{G}}$.

Note that the unperturbed `Linchpin` process utilizes just a single irreversible rule, yet provides the strongest possible form of performance guarantee. When the complement of this irreversible rule is introduced as a perturbation, a performance guarantee in the form of stochastic stability remains.

parameter	value	comment
N	14	total number of parts
$V_{\hat{G}}$	$\{1,2,3,4\}$	target has four parts
$E_{\hat{G}}$	$\{12,13,14\}$	see Figure 7
G_0	$(\{1,2,\dots,14\}, \emptyset, \ell_0 = 0)$	standard i.c.
ϵ	.01	disassembly term
a_r	1	nominal rule prob.
T	10,000	iterations per sim
F	uniform	agents selected uniformly
n	100	total number of sims

TABLE I
PARAMETERS FOR SIMULATIONS OF SELF-ASSEMBLY ALGORITHMS.

VI. CONSERVATISM OF COMPLETING RULES

The `Linchpin` gives unique final labels. However, unlike `Singleton` the states are not recoverable. That is, the correct labels (up to a label-preserving isomorphism) cannot always be directly inferred from the unlabeled subgraph. The implication is that the agents' states are not auxiliary. Each agent's behavior depends on more than just the structure of the assembly that it is participating in. The `Linchpin` algorithm will, in general, produce several rules of the form $\{0 \rightleftharpoons x - y\}$ with different x, y for each rule. Consequently, the labels of the resulting subgraphs cannot be inferred from the associated unlabeled subgraphs.

While `Singleton` has both of the aforementioned features, it only gives an asymptotically maximum yield (in $|V_{G_0}|$). A maximum yield is achieved by `Linchpin`, but the feature of internal states being derivable from the unlabeled graph is sacrificed. It is an open question whether any reversible algorithm obeying the communication constraints can satisfy both desiderata and give maximum yields. However, it turns out that if such an algorithm exists, it cannot exploit the notion of a completing rule. The proof is by way of a counterexample and is omitted herein for the sake of brevity, but can be found in [27].

Theorem 6.1: Any algorithm that gives reversible, binary constructive/deconstructive rule sets with completing rules must, for some target trees, either introduce states that cannot be determined from the unlabeled graph or give complete assemblies with non-unique states.

VII. SIMULATIONS

Simulations results are presented to compare performance between algorithms and to comment on transient behavior. Table I summarizes the parameters used in the simulations.

Simulations were run 100 times for each of the four algorithms: `Linchpin`, `Singleton`, non-reversible `Linchpin` (i.e. $\epsilon = 0$), and non-reversible `Singleton`. The non-reversible `Singleton` algorithm requires a larger rule set that we do not describe here. The rule probabilities $\mathcal{R}(r) = a_r = 1$ were used in each algorithm except for the rules in `Linchpin` and `Singleton` that depend on ϵ . The results of the simulations are displayed in Figure 10, after averaging over the 100 simulations for each algorithm.

The maximum yield for this simulation is three assemblies. The set of states with three complete assemblies is invariant under the two non-reversible processes. This is consistent

with the results of the simulation—all runs for both methods increase monotonically to the maximum yield within 2000 iterations and remain there. Since the two reversible processes do not exhibit such an invariance, both frequently contain fewer than three complete assemblies throughout the simulation. In the case of `Singleton`, states with three complete assemblies were observed less often than states with fewer complete assemblies. Recall that, from Theorem 4.2, no more than $(|V_{\hat{G}}| - 1)^2 = 9$ agents should fail to participate in complete assemblies in stochastically stable states of `Singleton`. Based on this bound only one complete assembly is expected for this application of `Singleton`. The fact two or three assemblies are more frequently observed highlights the conservatism of Theorem 4.2 as a bound on performance. A tighter bound can be constructed as a function of N and $|V_{\hat{G}}|$. When N is an integer multiple of $|V_{\hat{G}}|$, stochastically stable states all have maximum yield and Corollary 4.1 is the relevant bound. Theorem 4.2 is a worst-case bound based on Theorem 4.1. When $N \gg |V_{\hat{G}}|$ the difference between the worst-case and N -dependent bounds are small. Since $N = 14$ is not very large, it is readily apparent that the bound is not satisfied with equality. It is straightforward to verify that Theorem 4.1 implies that, in this example, all of `Singleton`'s stochastically stable states have at least two complete assemblies. Also, the minimum number of complete assemblies among stochastically stable states is not the only number we should expect to regularly observe. In our simulations, observation of three complete assemblies was nearly as likely as observation of two. The theory of stochastic stability says only that as ϵ goes to zero we should expect the states that are not stochastically stable to be observed with vanishing frequency. Some stochastically stable states can be much more frequently observed than others. This is why characterization of the set of stochastically stable states does not always give a tight bound on performance.

The proportion of time with each number of complete assemblies and the long-run average number of complete assemblies are summarized for each algorithm in table II.

VIII. DISCUSSION

A stochastic system framework was introduced for comparing the performance of different rule sets. Attention was restricted to binary constructive and deconstructive rules. Reversibility, recoverable states, and unique final states were additional requirements considered. The `Singleton` algorithm synthesizes rules for any connected acyclic target and provides a performance guarantee in the form of a bound on the number of reject assemblies among stochastically stable states. The `Linchpin` algorithm synthesizes rules for any connected acyclic target and provides a guaranteed maximum yield in the form of stochastic stability, so long as non-recoverable states are permitted. The maximum yield can be made an invariant of the system if even one irreversible rule is allowed. Both algorithms do not require any purely communicative rules, illustrating the possibility of self-assembly where such communication is difficult or impossible. Furthermore, existing algorithms that use communicative rules produce rule sets with greater cardinality than ours.

Fig. 10. The maximum yield of three is eventually reached and maintained in all simulations of the two non-reversible processes. For Lynchpin the system lingers around three, while for Singleton it lingers between two and three.

algorithm	0	1	2	3	mean
Singleton	.008	.032	.596	.364	2.316
Non-reversible Singleton	.000	.001	.002	.997	2.996
Lynchpin	.000	.001	.035	.965	2.964
Non-reversible Lynchpin	.000	.001	.001	.998	2.997

TABLE II

PROPORTION OF RUNNING TIME WITH EACH POSSIBLE NUMBER OF COMPLETE ASSEMBLIES.

The matter of whether or not a stronger performance guarantee can be made when recoverable states and unique final states are required remains an open question. We have seen some success in simulations by choosing different resistances for the deconstructive rules in the Singleton process so as to reduce the relative probability of disassembling more developed assemblies. Nevertheless, a rigorous analysis of such processes and an algorithm for finding these resistances for general $\hat{G}$ remain elusive. The convergence rates (mixing times) for the proposed algorithms are also unknown.

APPENDIX

STOCHASTIC STABILITY

Let $\mathcal{M}^\epsilon$ be an irreducible and aperiodic Markov chain transition matrix over a finite set of states Z for each $\epsilon \in (0, \bar{\epsilon}]$. If for each $z, z' \in Z$ we have

$$\lim_{\epsilon \rightarrow 0} \mathcal{M}_{z,z'}^\epsilon = \mathcal{M}_{z,z'}^0,$$

for some Markov chain $\mathcal{M}^0$ over Z , and

$$0 < \lim_{\epsilon \rightarrow 0} \frac{\mathcal{M}_{z,z'}^\epsilon}{\epsilon^{r(z,z')}} < \infty,$$

for some $r(z, z') \geq 0$ whenever $\mathcal{M}_{z,z'}^\epsilon > 0$ for some $\epsilon > 0$ then $\mathcal{M}^\epsilon$ is a *regular perturbed Markov process*. We call $\mathcal{M}^0$ the unperturbed process. If $\mathcal{M}_{z,z'}^\epsilon = 0$ for all ϵ , then we define $r(z, z') = \infty$. It is straightforward to see that $\mathcal{P}_{m,n}^\epsilon$ is a regular perturbed Markov process, with $\mathcal{P}_{m,n}^0$ being the reducible Markov chain obtained by substituting $\epsilon = 0$.

Let $\mu(\mathcal{M}^\epsilon)$ be the unique stationary distribution associated with $\mathcal{M}^\epsilon$, a state $z \in Z$ is *stochastically stable* if

$$\lim_{\epsilon \rightarrow 0} \mu_z(\mathcal{M}^\epsilon) > 0.$$

In order to characterize the stochastically stable states of $\mathcal{P}_{m,n}^\epsilon$, we will make use of the theory of resistance trees [28]. Let $R_1, \dots, R_J \subset Z$ be the recurrent communication classes of $\mathcal{M}^0$. Given two recurrent communication classes R_i and R_j , let $\{z_0, z_1, \dots, z_K\}$ be a path satisfying $z_0 \in R_i$ and $z_K \in R_j$. We call the quantity $\sum_{k=0}^{K-1} r(z_k, z_{k+1})$ the *resistance* of the path. With slight abuse of notation we define r_{ij} to be the *least resistance* among all such paths.

Consider a graph G whose vertex set is the set of recurrent communication classes. An R_i -tree T is a spanning tree in G such that for any vertex $R_j, j \neq i$ there is a unique directed

path from R_j to R_i . Define

$$\gamma(R_i) = \min_{T \in \mathcal{T}_{R_i}} \sum_{(R_j, R_k) \in T} r_{jk},$$

where $\mathcal{T}_{R_i}$ is the set of all R_i trees in G , which we refer to as the *stochastic potential* of R_i . The following theorem [28] characterizes exactly the set of stochastically stable states.

Theorem A.1: Let $\mathcal{M}^\epsilon$ be a regular perturbed Markov process and let $R_1, \dots, R_J$ be the recurrent communication classes of the unperturbed process $\mathcal{M}^0$. Then the stochastically stable states are precisely those states contained in the recurrent communication classes with minimum stochastic potential.

SINGLETON ALGORITHM WALKTHROUGH

The Singleton algorithm is illustrated for the target graph of Example 4.1. The target graph is:

$$V_{\hat{G}} = \{1', 2', 3', 4', 5', 6'\}$$

$$E_{\hat{G}} = \{1'2', 1'3', 1'4', 1'5', 5'6'\}.$$

Since the labels generated by Singleton are numerical, the “’” superscript is added to distinguish the vertex numbering for the target graph and the labels generated by Singleton.

Initiate with the call `Singleton( $V_{\hat{G}}, E_{\hat{G}}, 1', 0$ )`.

- The “working” root vertex is $k = 1'$ with neighbors $\{2', 3', 4', 5'\}$. $s = 0$. $\bar{s} = 0$.
- The rules “ $0 \ 0 \Rightarrow 1 - 2$ ” are created, with label 1 associated with $1'$ and label 2 associated with $2'$. $s = 2$. $\bar{s} = 1$.
- Singleton is called with the root vertex $k = 2'$, which has no neighbors, and so no new rules are returned.
- The rules “ $1 \ 0 \Rightarrow 3 - 4$ ” are created, with label 3 associated with $1'$ and label 4 associated with $3'$. $s = 4$. $\bar{s} = 3$.
- Singleton is called with the root vertex $k = 3'$, which has no neighbors, and so no new rules are returned.
- The rules “ $3 \ 0 \Rightarrow 5 - 6$ ” are created, with label 5 associated with $1'$ and label 6 associated with $4'$. $s = 6$. $\bar{s} = 5$.
- Singleton is called with the root vertex $k = 4'$, which has no neighbors, and so no new rules are returned.
- The rules “ $5 \ 0 \Rightarrow 7 - 8$ ” are created, with label 7 associated with $1'$ and label 8 associated with $5'$. $s = 8$. $\bar{s} = 7$.
- Singleton is called with the root vertex $k = 5'$, graph $V = \{5', 6'\}$, $E = \{5'6'\}$, and $s = 8$.
- The rules “ $8 \ 0 \Rightarrow 9 - 10$ ” are created, with label 9 associated with $5'$ and label 10 associated with $6'$.
- Singleton is called with the root vertex $k = 6'$, which has no neighbors, and so no new rules are returned.
- Singleton completes rule creation with $k = 5'$ as the root vertex.
- Singleton completes rule creation with $k = 1'$ as the root vertex.

LINCHPIN ALGORITHM WALKTHROUGH

The Linchpin algorithm is illustrated for the target graph of Example 4.1. The target graph is:

$$V_{\hat{G}} = \{1', 2', 3', 4', 5', 6'\}$$

$$E_{\hat{G}} = \{1'2', 1'3', 1'4', 1'5', 5'6'\}.$$

As with the Singleton walkthrough, the “’” superscript is added to distinguish the vertex numbering for the target graph and the labels generated by Linchpin.

Initiate with the call `Linchpin( $V_{\hat{G}}, E_{\hat{G}}, 1', 0$ )`.

- The “working” root vertex is $k = 1'$ with neighbors $\{2', 3', 4', 5'\}$.
- Vertices $\{2', 3', 4'\}$ have no other neighbors. There are no rules generated for them at this stage. Set $s_{2'} = s_{3'} = s_{4'} = 0$.
- Linchpin is called with the root vertex $k = 5'$, graph $V = \{5', 6'\}$, $E = \{5'6'\}$, and $s = 0$.
- The rules “ $0 \ 0 \Rightarrow 1 - 2$ ” are created, with label 1 associated with $6'$ and label 2 with $5'$. $s_{5'} = s = 2$.
- The rules “ $0 \ 0 \Rightarrow 3 - 4$ ” are created, with label 3 associated with $2'$ and label 4 associated with $1'$. $s = 4$.
- The rules “ $0 \ 4 \Rightarrow 5 - 6$ ” are created, with label 5 associated with $3'$ and label 6 with $1'$. $s = 6$.
- The rules “ $0 \ 6 \Rightarrow 7 - 8$ ” are created, with label 7 associated with $4'$ and label 8 associated with $1'$. $s = 8$.
- The (completion) rule and its complement, “ $2 \ 8 \Rightarrow 9 - 10$ ” are created, with label 9 associated with $5'$ and label 10 associated with $1'$. $s = 10$.

ACKNOWLEDGEMENT

We thank Eric Klavins, Magnus Egerstedt, Martha Grover, Yuzhen Xue, Nikolaus Correll, Vijeth Rai, and Dustin Reishus for helpful discussions.

REFERENCES

- [1] J. von Neumann, “Re-evaluation of the problems of complicated automata problems of hierarchy and evolution,” Dec. 1949, fifth Lecture, University of Illinois.
- [2] V. Claus, H. Ehrig, and G. Rozenberg, Eds., *Graph-Grammars and Their Application to Computer Science and Biology, International Workshop, Bad Honnef, October 30 - November 3, 1978*, ser. Lecture Notes in Computer Science, vol. 73. Springer, 1979.
- [3] M. Fox and J. Shamma, “Communication, convergence, and stochastic stability in self-assembly,” in *49th IEEE Conference on Decision and Control*, Dec. 2010.
- [4] —, “Self-assembly for maximum yields under constraints,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.
- [5] E. Klavins, “Automatic synthesis of controllers for distributed assembly and formation forming,” in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2002.
- [6] E. Klavins, R. Ghrist, and D. Lipsky, “A grammatical approach to self-organizing robotic systems,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 949–962, Jun. 2006.
- [7] E. Klavins, S. Burden, and N. Napp, “Optimal rules for programmed stochastic self-assembly,” in *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.
- [8] N. Napp, S. Burden, and E. Klavins, “Setpoint regulation for stochastically interacting robots,” *Auton. Robots*, vol. 30, no. 1, pp. 57–71, Jan. 2011. [Online]. Available: <http://dx.doi.org/10.1007/s10514-010-9203-2>
- [9] E. Klavins, “Programmable self-assembly,” *Control Systems Magazine*, vol. 24, no. 4, pp. 43–56, Aug. 2007.

- [10] Y. Xue and M. A. Grover, "Optimal design for active self-assembly system," in *Proceedings of the 2011 American Control Conference*, San Francisco, CA, June 2011.
- [11] J. S. Baras, T. Jiang, and P. Hovareshti, "Coalition formation and trust in collaborative control," in *Proceedings of the European Control Conference 2009*, Budapest, Hungary, August 2009.
- [12] H. E. Samad and M. Khammash, "Stochastic stability and its application to the analysis of gene regulatory networks," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 3, Dec. 2004, pp. 3001–3006.
- [13] K. Kotay and D. Rus, "Generic distributed assembly and repair algorithms for self-reconfiguring robots," in *IEEE Intl. Conf. on Intelligent Robots and Systems*, Sendai, Japan, 2004.
- [14] K. Fujibayashi, R. Hariadi, S. Park, E. Winfree, and S. Murata, "Toward Reliable Algorithmic Self-Assembly of DNA Tiles: A Fixed-Width Cellular Automaton Pattern," *Nano Letters*, vol. 8, no. 7, pp. 1791–1797, 2008.
- [15] D. Reishus, "On the mathematics of self-assembly," Ph.D. dissertation, University of Southern California, 2009.
- [16] B. Salemi, M. Moll, and W. Shen, "Superbot: A deployable, multifunctional, and modular self-reconfigurable robotic system," in *Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Oct. 2006.
- [17] N. Ayanian, P. White, A. Halasz, M. Yim, and V. Kumar, "Stochastic control for self-assembly of xbots," in *ASME Mechanisms and Robotics Conference*, August 2008.
- [18] F. Hou and W. Shen, "Mathematical foundation for hormone-inspired control for self-reconfigurable robotic systems," in *Proc. 2006 IEEE Intl. Conf. on Robotics and Automation*, May 2006.
- [19] R. Nagpal, "Programmable self-assembly using biologically-inspired multiagent control," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, July 2002.
- [20] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 42–52, Mar. 2007.
- [21] M. Boncheva, D. Bruzewicz, and G. Whitesides, "Millimeter-scale self-assembly and its applications," *Pure Appl. Chem.*, vol. 75, pp. 621–630, 2003.
- [22] V. Rai, A. van Rossum, and N. Correll, "Self-assembly of modular robots from finite number of modules using graph grammars," in *In Proceedings of the International Conference on Intelligent Robots and Systems*. San Francisco, CA: IEEE/RSJ, 2011.
- [23] D. C. Rapaport, "Role of reversibility in viral capsid growth: A paradigm for self-assembly," *Physical Review Letters*, vol. 101, Oct. 2008.
- [24] U. Majumder, J. Reif, and S. Sahu, "Stochastic analysis of reversible self-assembly," *Journal of Computational and Theoretical Nanoscience*, vol. 5, no. 7, pp. 1289–1305, Jul. 2008.
- [25] H. Young, *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*. Princeton University Press, 1998.
- [26] M. Freidlin and A. Wentzell, *Random perturbations of dynamical systems*, ser. Grundlehren der mathematischen Wissenschaften. Springer, 1998, no. v. 260. [Online]. Available: <http://books.google.com/books?id=0yE74YEXpWEC>
- [27] M. Fox, "Distributed learning in large populations," Ph.D. dissertation, Georgia Institute of Technology, August 2012.
- [28] H. Young, "The evolution of conventions," *Econometrica*, vol. 61, no. 1, pp. 57–84, January 1993.

Jeff S. Shamma is a Professor of Electrical Engineering in the King Abdullah University of Science and Technology (KAUST) and the Julian T. Hightower Chair in Systems & Control in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He received a BS in Mechanical Engineering from Georgia Tech in 1983 and a PhD in Systems Science and Engineering from the Massachusetts Institute of Technology in 1988. He held faculty positions at the University of Minnesota, University of Texas-Austin, and University of California-Los Angeles. Shamma is a recipient of the NSF Young Investigator Award (1992), the American Automatic Control Council Donald P. Eckman Award (1996), and the Mohammed Dahleh Award (2013), and he is a Fellow of the IEEE (2006). He is currently an Associate Editor for the *IEEE Transactions on Cybernetics* (2009-present) and *Games* (2012-present) and a Senior Editor for the *IEEE Transactions on Control of Network Systems* (2013-present). Shamma's research is in the general area of feedback control and systems theory. His most recent research has been in decision and control for distributed multiagent systems and the related topics of game theory and network science, with applications to cyberphysical and societal network systems.

Michael J. Fox is a Quantitative Research Associate at Tower Research Capital, an investment management firm. He received a BS in Electrical Engineering from The Cooper Union for the Advancement of Science and Art in 2008 and a PhD in Electrical and Computer Engineering from the Georgia Institute of Technology in 2012. He is a recipient of the Abraham J. Pletman Memorial Prize (2008) and an NDSEG Graduate Fellowship (2009). Dr. Fox's research investigates the theory of learning in games as well as applications to language, robotics, and

congestion management.