Optimal Rules for Programmed Stochastic Self-Assembly

Eric Klavins Samuel Burden Nils Napp

Electrical Engineering
University of Washington
Seattle, WA 98195

***Abstract*— We consider the control of programmable self-assembling systems whose dynamics are governed by stochastic reaction-diffusion dynamics. In our system, particles may decide the outcomes of reactions initiated by the environment, thereby steering the global system to produce a desired assembly type. We describe a method that automatically generates a program maximizing yield based on tuning the rates of experimentally determined reaction pathways. We demonstrate the method using theoretical examples and with a robotic testbed. Finally, we present, in the form of a graph grammar, a communication protocol that implements the generated programs in a distributed manner.**

I. Introduction

Self-assembly is a phenomenon in which a *soup* of particles spontaneously arrange themselves into a coherent structure. Self-assembly is ubiquitous in nature. For example, virus capsids, cell-membranes and tissues are all self-assembled from smaller components in a completely distributed fashion. Self-assembly is beginning to find its way into engineering, harnessed using a variety of technologies such as DNA [1], [2], PDMS [3], MEMs [4] and robots [5], [6], [7].

Self-assembly comes in, roughly, two flavors: passive and active. In passive self-assembly, particles interact according to geometry or surface chemistry and tend toward a thermodynamic equilibrium at which the system is assembled. For example, phospholipids stick to each other along hydrophobic regions to form membranes. In active self-assembly, the particles can somehow decide in what interactions to partake. For example, proteins in cells may undergo *conformational switching* that changes the outcomes of their subsequent interactions in very specific ways.

This paper is about the active model. The systems we consider are still dominated by thermodynamics, so that structures form and decay according to natural rates. The particles merely steer the dynamics of the system by disallowing or slowing some of the reaction pathways in which they participate. We believe this model may apply to a wide variety of phenomena in which reaction pathway tuning occurs. For example, directed evolution is routinely used to tune metabolic pathways [8].

Besides exploring a number of theoretical models of programmed self-assembly, we have built a self-assembling system consisting of a number of very simple robotic parts (Figure 1) [5]. When mixed on an air-hockey table, the parts randomly collide and bind to each other. Once bound, the parts may communicate with each other and, at some future time, may decide to detach from each other. Besides that, the parts simply diffuse through the environment. Nevertheless, we are able to program the system to form pre-specified shapes, as illustrated in Figure 2.

This work is partially supported by NSF Grant #0347955: *CAREER: Programmed Robotic Self-Assembly*. S. Burden is also partially supported by a *Mary Gates Undergraduate Research Training Fellowship*.

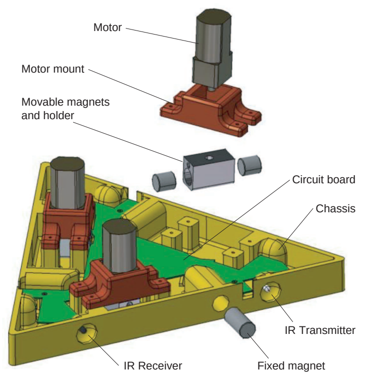


Fig. 1. The programmable part mechanism includes low power magnetic latches, infrared communications, and an on-board micro-controller [5]. The parts do not move themselves. They float on an air table and are mixed randomly by airjets along the perimeter.

Here we explain how to program particles so that the yield of a desired assembly type at equilibrium is maximized. We describe a model of the system based on chemical kinetics that includes the effect of programming the interactions. We then pose a general optimization problem in which maximizing a certain function of the equilibrium state of the Markov Process describing the system results in optimal self-assembly programs. Finally, we describe how the results of the optimization (a vector of probabilities) can be encoded into a communications protocol for the robotic parts. This extends our earlier work from the nondeterministic setting, wherein we guarantee only that the desired assembly is reachable and stable, to the stochastic setting. We give several examples, including the application of the ideas to the programmable parts system.

II. Related Work

The present paper owes much to the work of Hosokawa, Shimoyama and Miura who describe their macroscale self-assembling system using models from chemical kinetics [9]. Their system consisted of passive triangular parts in a vertical shaker, whereas our parts are programmable. The modeling techniques in the present paper are applicable to robots,

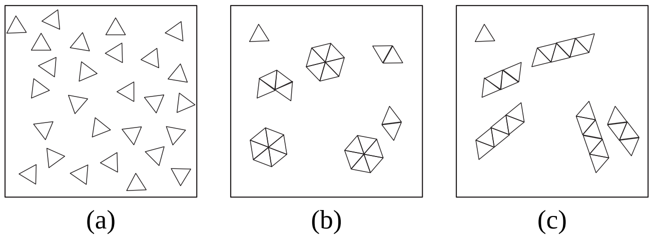


Fig. 2. (a) The programmable parts are initially unattached. (b) After random mixing, the parts form hexagons by rejecting certain binding events and accepting others according to their programming. (c) The parts can be reprogrammed to form other assembly types, such as chains. This paper addresses the problem of programming the parts to maximize the yield of the desired assembly type.

built by several groups, that either float on an air table [10], [5] or even are suspended in oil [7]. The design of such robots borrows heavily from other types of modular robots [11], [12]. We also believe that these ideas will be applicable to micro- and nano-scale self-assembly problems [3], [1]. This work is aimed at building a comprehensive dynamical systems model and programming discipline for these systems and builds upon the grammatical approach described in Section VII and in [13], [14]. The combination of standard ideas from chemical kinetics [15], the notion of programmable reactions (Section V) and optimization (Section VI) is new, to the best of our knowledge. Besides the grammatical approach, the most relevant work toward understanding self-assembly "programs" in a stochastic setting is nucleic acid design [16], where the free-energy landscape associated with DNA hybridization reactions is engineered. The present work differs in that we suppose that local interactions are controllable, whereas in the DNA work the initial oligo sequences are the programmable elements.

III. The Stochastic Model of Self-Assembly

A. Macrostates and Reactions

A *component* is a connected group of particles. The self-assembly of a component occurs when two smaller components combine. We denote the set of all *component types* in a system by $\mathcal{C} = \{C_1, C_2, C_3, ...\}$. The number of component types may be either finite or countably infinite. A *macrostate* describes the number of each type of component in the system at a given time. Formally, a macrostate $\mathbf{v}$ is a function from $\mathcal{C}$ to the natural numbers. We write macrostates as vectors, for example,

$$\mathbf{v} = (10 \;\; 5 \;\; 2 \;\; 0 \;\; ... \;\;)^T \tag{1}$$

to denote the state in which there are 10 components of type C_1, 5 component of type C_2, 2 components of type C_3 an no components of any other type.

A *reaction* is a vector that describes an assembly or disassembly event as in, for example,

$$\mathbf{a} = (-1 \;\; -1 \;\; 1 \;\; 0 \;\; ... \;\;)^T, \tag{2}$$

which describes the reaction type $C_1 + C_2 \rightharpoonup C_3$. The *result* of a reaction $\mathbf{a}$ in the macrostate $\mathbf{v}$ is

$$\mathbf{v}' = \mathbf{v} + \mathbf{a}$$

when all the entries of $\mathbf{v}'$ are non-negative. In this case, $\mathbf{a}$ is said to be *applicable* to $\mathbf{v}$.

When a reaction describes the combination of two components, it is called a *forward* reaction. When it describes the degradation of a component into smaller components, then it is called a *reverse reaction*. The *inverse* of a forward reaction $\mathbf{a}$ is $\mathbf{a}^{-1} = -\mathbf{a}$. The sets of forward and reverse actions applicable to a macrostate $\mathbf{v}$ are denoted $\mathcal{F}(\mathbf{v})$ and $\mathcal{R}(\mathbf{v})$, respectively.

The matrix $\mathbf{A} = (\mathbf{a}_1 \; ... \; \mathbf{a}_k)$ whose columns are all the forward reaction types in a given system is called the *forward reaction matrix* (or stoichiometric matrix). The matrix $-\mathbf{A}$ is the *reverse reaction matrix*.

B. Rates

Each reaction $\mathbf{a}$ has associated with it a *stochastic rate constant* $k(\mathbf{a})$ that describes the rate at which the reaction occurs given that the reactants (those components with negative entries in $\mathbf{a}$) have encountered each other. Typically, these rates must be measured from experiments (as, for example, we describe in Section IV-B).

The *multiplicity* $M(\mathbf{v}, \mathbf{a})$ of the reaction $\mathbf{a}$ in macrostate $\mathbf{v}$ is the number of ways in which $\mathbf{a}$ can occur in $\mathbf{v}$. For example, the multiplicity of the reaction $\mathbf{a}$ defined by Equation (2) evaluated in the macrostate $\mathbf{v}$ defined by Equation (1) is $\mathbf{v}(1)\mathbf{v}(2) = 10 \cdot 5 = 50$, there being 50 ways in which a component of type C_1 and a component of type C_2 can be chosen in $\mathbf{v}$ to react according to $\mathbf{a}$. If $\mathbf{a}$ is not applicable to $\mathbf{v}$, then $M(\mathbf{v}, \mathbf{a})$ is zero.

The *rate* of the reaction $\mathbf{v} \rightharpoonup \mathbf{v} + \mathbf{a}$ is defined by

$$K(\mathbf{v}, \mathbf{a}) = k(\mathbf{a}) M(\mathbf{v}, \mathbf{a}).$$

The vector-valued functions $\mathbf{K}_f$ and $\mathbf{K}_r$ defined by

$$\mathbf{K}_f(\mathbf{v}) = (K(\mathbf{v}, \mathbf{a}_1), ..., K(\mathbf{v}, \mathbf{a}_k))^T$$

and

$$\mathbf{K}_r(\mathbf{v}) = (K(\mathbf{v}, \mathbf{a}_1^{-1}), ..., K(\mathbf{v}, \mathbf{a}_k^{-1}))^T,$$

where $\mathbf{a}_1, ..., \mathbf{a}_k$ are the forward reactions of a given system, are called the *forward* and *reverse rate functions* of the system, respectively.

C. Dynamics

A *system*, given by an initial macrostate $\mathbf{v}_0$, a forward reaction matrix $\mathbf{A}$ and the forward and reverse rate functions $\mathbf{K}_f$ and $\mathbf{K}_r$, describes a stochastic system. We interpret a system as either a continuous-time, discrete-state Markov process or as a set of mass action kinetics equations. The former is useful for systems with relatively small numbers of particles and is also useful for obtaining the rates of a system experimentally [6]. The latter is useful for systems with large numbers of particles and is useful for evaluating the average performance of a system.

a) Markov Process: The interpretation of the system as a Markov Process uses the *Master Equation* from chemical kinetics [17]. The states are the possible macrostates $\mathbf{v}$, which we enumerate using a simple (although computationally intensive) algorithm: Let $\mathbf{v}_0$ be the initial macrostate $(n \;\; 0 \;\; 0 \;\; ...)^T$. Apply each action $\mathbf{a}_i$ in the system to $\mathbf{v}_0$ to obtain new macrostates $\mathbf{v}_1 ... \mathbf{v}_k$. To each of those, apply all the actions again to obtain a (possibly) new set of macrostates and so on. Continue this process until all N macrostates have been enumerated. In the systems we examine here, reactions do not create new mass, and thus this procedure will terminate. It is useful to define, $\mathbf{S} = (\mathbf{v}_0 \ \mathbf{v}_1 \ ... \ \mathbf{v}_N)$, the $|\mathcal{C}| \times N$ dimensional matrix of reachable macrostates. In general, there are $O(2^n)$ macrostates for a system with n particles and $O(n)$ component types. Nevertheless, this formulation is quite useful for understanding small systems or fragments of larger systems.

Let $x_i(t)$ be the probability that the system is in macrostate $\mathbf{v}_i$ at time t. The rate at which x_i transitions to x_j is given by

$$Q_{i,j} = \sum_{\mathbf{v}_j = \mathbf{v}_i + \mathbf{a}} K(\mathbf{v}_i, \mathbf{a}) + \sum_{\mathbf{v}_j = \mathbf{v}_i - \mathbf{a}} K(\mathbf{v}_i, -\mathbf{a})$$

where $\mathbf{a}$ ranges over all forward reactions applicable to $\mathbf{v}_i$ and which result in $\mathbf{v}_j$. The diagonal elements $Q_{i,i}$ are the negative sum of the elements $Q_{i,k}$ for $k \neq i$. The average behavior of the system is then given by

$$\dot{\mathbf{x}} = \mathbf{Q}^T \mathbf{x},$$

which is Kolmogorov's Forward Equation [18, pp. 85-86]. A steady state distribution of the system is obtained by solving $\mathbf{Q}^T \mathbf{x} = \mathbf{0}$ and is denoted $\mathbf{x}(\infty)$. Because each reaction is reversable, $\mathbf{x}(\infty)$ is typically unique in these problems. The product

$$\mathbf{v} = \mathbf{S}\mathbf{x}$$

gives the expected number of each component type as a function of time.

The dynamics $\dot{\mathbf{x}}$ described by $\mathbf{Q}^T$ should be thought of as the average behavior of many experiments. Individual trajectories of the system must be obtained through simulation, using *Gillespie's* method [17]. In this paper, we will be primarily concerned with the steady state behavior $\mathbf{x}(\infty)$ and leave the control of the transient behaviors to a later paper.

b) Mass Action Kinetics: In the interpretation of the system using mass action kinetics, we suppose that $\mathbf{v}(i) \in \mathbb{R}$ represents the concentration of component i in the system. We also typically use $M(\mathbf{v}, \mathbf{a}) = i^2$ instead of $i(i-1)$ for the multiplicity of forward reactions involving the same component type i. The equation describing the behavior is

$$\dot{\mathbf{v}} = \mathbf{A} \cdot (\mathbf{K}_f(\mathbf{v}) - \mathbf{K}_r(\mathbf{v})).$$

This is, in general, a nonlinear equation for which only numerical solutions can be obtained. It is interesting to note that finding the steady state of the mass action kinetics requires solving the nonlinear $\dot{\mathbf{v}} = \mathbf{0}$, while finding the steady state of Kolmogorov's equation requires finding the null-space of the high-dimensional linear operator $\mathbf{Q}^T$. Both yield the same information about the steady state.

IV. Natural Systems

The components in $\mathcal{C}$ we consider are each composed of some number of parts. When two components interact, they do so via one part from each component, and these two parts can locally *decide* whether or not to bind according to their programming. In this section, we consider *natural* systems in which interacting parts *always* decide to bind.

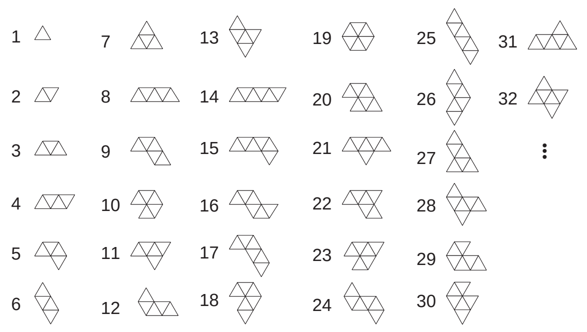


Fig. 3. The indexing scheme for the smaller components that can form in the programmable parts system.

We consider two examples. The first, a simple model of polymerization, is primarily for purposes of illustration. The second, the *programmable parts*, deals with our testbed robots.

A. Polymerization

The first example deals with simple labeled acyclic graphs with degree at most two. Thus, the components of the system are

$$\mathcal{C}_1 = \{P_1, P_2, P_3, P_4, ... P_n\},$$

where P_i denotes a path of length i. We suppose there are n parts in total, so the largest object that can be built is P_n. Each forward reaction either joins two paths ($P_i + P_j \rightharpoonup P_{i+j}$) or breaks a path ($P_{i+j} \rightharpoonup P_i + P_j$). For example, assuming there are six parts (or that only components up to size six are permitted), the reaction matrix is

$$\mathbf{A} = \begin{pmatrix} -2 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & -2 & -1 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 & -2 \\ 0 & 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

A reasonable model for the rates is to suppose that each forward action $\mathbf{a}$ occurs at the rate $k(\mathbf{a})$, and its reverse occurs at the rate $k(\mathbf{a}^{-1}) = k(\mathbf{a})e^{-\varepsilon}$, where $\varepsilon > 0$ is the energy of a bond. To construct examples, we will choose the forward rates randomly.

B. The Programmable Parts

The components in the programmable parts system are connected triangular sub-tilings of the plane. We index the smaller components for easy reference, as shown in Figure 3.

To determine the rates for reactions between programmable part components, we use a high-fidelity mechanics-based simulation of the system [6] and fit the stochastic process described by the Master Equation to the data. We repeat the essential details here for completeness.

We fix the average kinetic energy of the parts (i.e. the *temperature*) at $K_{ave} = 5 \times 10^{-4} J$, which is the kinetic energy measured from experiments using the actual robots

1	+	2	$\rightharpoonup$	3	0.00679 ± 0.00028
1	+	1	$\rightharpoonup$	2	0.0112 ± 0.0006
1	+	5	$\rightharpoonup$	10	0.0110 ± 0.0012
1	+	3	$\rightharpoonup$	6	0.00261 ± 0.00025
2	+	2	$\rightharpoonup$	6	0.00304 ± 0.00034
2	+	3	$\rightharpoonup$	10	0.00638 ± 0.00082
2	+	5	$\rightharpoonup$	19	0.00182 ± 0.00050
3	+	3	$\rightharpoonup$	19	0.00118 ± 0.00020
1	+	10	$\rightharpoonup$	19	0.00187 ± 0.00090
6	$\rightharpoonup$	1	+	3	0.000951 ± 0.00024
2	$\rightharpoonup$	1	+	1	0.000133 ± 0.000130
4	$\rightharpoonup$	1	+	3	0.00110 ± 0.00021
4	$\rightharpoonup$	2	+	2	0.00111 ± 0.00035

Fig. 4. Some of the 272 kinetic rate constants we measured for the programmable part system with 12 parts, $\rho = 5\text{parts}/m^2$ and $K_{ave} = 5 \times 10^{-4}$ J. The initial macrostate for each approximation is chosen so that there are an approximately equal number of the *forward* reactants (see Figure 3 for a listing of the reactant types). The errors in the rate constants for the reverse reactions are in general larger because we observed fewer reverse reactions. Note that the error bars do not account for errors in the measurement of the physical properties of the programmable parts.

in the laboratory. We also fix a density of $\rho = 5$ parts$/m^2$ that (1) approximately matches the physical testbed and (2) comes from a parameter regime where the reaction-diffusion model is valid [6], [17]. The simulations use $N = \rho A = 10$ parts.

As an example, suppose $\mathbf{a}$ is the reaction $1 + 2 \rightharpoonup 3$. To determine the rate $k(\mathbf{a})$ for a single part (component type 1) combining with a "dimer" (component type 2) to form a "trimer" (component type 3), we choose a macrostate of the form

$$\mathbf{v}_0 = (N_1 \ N_2 \ 0 \ 0 \ ...)^T$$

with N_1 single parts and N_2 dimers. We run n simulations from random initial positions and with the initial velocities chosen from a Gaussian distribution with mean equal to K_{ave}/m, where m is the mass of a part. As soon as a reaction occurs, in this case either $1+2 \rightharpoonup 3$ or $2 \rightharpoonup 1+1$, we restart the simulation with a new random initial condition. *We do not reset the time t when we restart the simulation.*

Suppose that the times at which a reaction $\mathbf{a}$ occurs are

$$\tau(\mathbf{a}) = (t_1, t_2, ..., t_r).$$

The hypothesis (from the assumption that the system is a Markov Process) is that the intervals $\Delta t_i = t_{i+1} - t_i$ are distributed according to a Poisson waiting process with mean $\lambda = 1/k(\mathbf{a})$. We thus arrive at the estimate

$$k(\mathbf{a}) \approx \frac{1}{N_1 N_2} \left(\frac{1}{\langle \Delta t \rangle} \pm \frac{std(\Delta t)}{\sqrt{n} \langle \Delta t \rangle^2} \right)$$

where $\langle \Delta t \rangle$ is the average waiting interval for the reaction and $std(\Delta t)$ is its standard deviation. Note that $N_1 N_2$ is the multiplicity $M(\mathbf{v}_0, \mathbf{a})$. Said differently, the approximate rate constant is the rate at which the reaction occurs starting at $\mathbf{v}_0$ divided by the multiplicity of the reaction in $\mathbf{v}_0$. From the same set of simulations, we also obtain an approximation of the rate $k(2 \rightharpoonup 1 + 1)$ from the times $\tau(2 \rightharpoonup 1 + 1)$.

In Figure 4 we list the rates obtained from simulation for a number of the most primitive reactions of the system. Each rate was obtained from between 500 and 1000 observed simulated reactions. Simulations of the stochastic system described by just the rates and using Gillespie's method [17] match the full-mechanics-based simulations of the programmable parts and experiments using the actual testbed robots.

V. Programmed Systems

In a programmed system, two interacting components may *decide* what to do after binding. For example, suppose the goal is to make copies of P_3 in the polymerization example. If a P_2 and a P_2 interact, they should temporarily form a P_4. Then the parts making up the P_4 should shed a P_1, leaving a P_3. Suppose that k_{com}, is the rate at which the parts decide and communicate what to do. Then the *natural* reaction $P_2 + P_2 \rightharpoonup P_4$ becomes the programmed reaction *pathway*

$$P_2 + P_2 \rightharpoonup P_4 \rightharpoonup P_3 + P_1$$

or simply

$$P_2 + P_2 \rightharpoonup P_3 + P_1,$$

when we assume that $k_{com} \gg k_f$. We describe a communications protocol that implements programmed reactions using the language of *graph grammars* [13] in Section VII.

We write $\widetilde{\mathbf{a}}$ for a reaction that has been programmed. In the above example, we have changed the fifth reaction in the polymerization example from

$$\mathbf{a}_5 = (0 \quad -2 \quad 0 \quad 1 \quad ...)^T$$

to

$$\widetilde{\mathbf{a}}_5 = (1 \quad -2 \quad 1 \quad 0 \quad ...)^T.$$

More generally, we may wish to consider all possible ways to break apart P_4. There are five possibilities:

$$\begin{array}{ll} 1: P_4 & 4: P_2 + 2P_1 \\ 2: P_3 + P_1 & 5: 4P_1 \\ 3: 2P_2 & \end{array}$$

Define $q_{4,j}$ to be the probability that the parts decide to use the jth option in the above list after the natural reaction $\mathbf{a}_5$ produces a P_4. The resulting programmed reaction can be written as

$$\begin{array}{rcl} \widetilde{\mathbf{a}}_5 & = & -(0\ 2\ 0\ 0\ 0\ \ldots)^T + q_{4,1}(0\ 0\ 0\ 1\ 0\ \ldots)^T \\ & + & q_{4,2}(1\ 0\ 1\ 0\ 0\ \ldots)^T + q_{4,3}(0\ 2\ 0\ 0\ 0\ \ldots)^T \\ & + & q_{4,4}(2\ 1\ 0\ 0\ 0\ \ldots)^T + q_{4,5}(4\ 0\ 0\ 0\ 0\ \ldots)^T \end{array}$$

where $\sum q_{4,j} = 1$. Each programmed action therefore has the form

$$\widetilde{\mathbf{a}_i} = \mathbf{a}_{i,0} + \sum_j q_{k,j} \mathbf{a}_{i,j}$$

where action $\mathbf{a}_i$ is assumed to produce component k, the index j ranges over the options of how to break-up component k, and $\mathbf{a}_{i,0}$ describes the components consumed by the reaction.

If $\mathbf{A}$ is the natural (un-programmed) reaction matrix then we write $\widetilde{\mathbf{A}}_\mathbf{q}$ for the reaction matrix programmed with the probabilities $\mathbf{q}$. Notice that the rates of the programmed actions are identical to the rates of the natural reactions.

Also notice that components still decay according to the natural reverse reactions in $-\mathbf{A}$. This is easily seen in the new equation describing the mass action kinetics

$$\dot{\mathbf{v}} = \widetilde{\mathbf{A}}_{\mathbf{q}} \cdot \mathbf{K}_f(\mathbf{v}) - \mathbf{A} \cdot \mathbf{K}_r(\mathbf{v}). \tag{3}$$

In the interpretation of the system as a Markov Process, we have that the matrix $\mathbf{Q}$ is now parameterized by the vector $\mathbf{q}$ and the averaged dynamics become

$$\dot{\mathbf{x}} = \mathbf{Q}(\mathbf{q})^T \mathbf{x}. \tag{4}$$

The off-diagonal elements of $\mathbf{Q}(\mathbf{q})$ are

$$Q(\mathbf{q})_{i,j} = \sum_{\mathbf{v}_j = \mathbf{v}_i + \mathbf{a}_{k,0} + \mathbf{a}_{k,l}} q_{p_k,l} K(\mathbf{v}, \mathbf{a}_k) + \sum_{\mathbf{v}_j = \mathbf{v}_i - \mathbf{a}} K(\mathbf{v}_i, -\mathbf{a})$$

where the first sum ranges over all k and l such that $\mathbf{v}_j$ is obtained from $\mathbf{v}_i$ by using the l^{th} option for breaking up the product C_{p_k} of forward reaction $\mathbf{a}_k$.

VI. Optimal Programs

We wish to find a probability vector $\mathbf{q}$ so that a cost J is optimized subject to the constraints $\sum_j q_{i,j} = 1$ and the dynamics. We will use the Markov Chain interpretation of the rates so that the resulting optimization problem is linear.

For the self-assembly problem, let

$$\mathbf{c} = (0 \ldots 0\ 1\ 0 \ldots 0)^T$$

be the $|\mathcal{C}| \times 1$ dimensional vector with zeros everywhere except in the mth place. To maximize the yield of component m we we maximize the function

$$J_{assem} = \mathbf{c}^T \mathbf{S} \mathbf{x}$$

subject to $\mathbf{Q}^T \mathbf{x} = 0$ and the constraints

$$\sum_j q_{i,j} = 1$$

for all i. This problem can be recast as a bilinear programming problem and solved (locally) with existing software [19]. The solution can be obtained in polynomial time in the number of probabilities in $\mathbf{q}$ and the dimension $N+1$ of $\mathbf{x}$, which unfortunately is usually quite large.

The optimal probabilities in Equation 3 determine the *usefulness* of each component in $\mathcal{C}$. Obviously, the desired component i should be formed after a natural reaction whenever possible, by detaching sub-components. The sub-components may need to be further dismantled, depending on the other rates. For example, suppose the result of a natural reaction $\mathbf{a}$ can be broken into the desired component i and another component j. It may be that the rates of forward reactions involving j are low or zero – so j is a *dead-end* component. In that case, component j should be broken-down further.

When the goal is to maximize the yield of the mth component, each reaction is programmed as follows. If the result of the natural reaction contains the mth component as a sub-component, then keep that component and release and dismantle the remaining components. If the result of the natural reaction does not contain the mth component, simply dismantle the result of the reaction. In the examples below, we only consider how to dismantle components whose size is less than or equal to the size of the mth component.

A. The Polymerization Example

Suppose the goal is to maximize the number of components of type P_4. Any reaction resulting in something larger than P_4 will be broken into a P_4 and some other component. Furthermore, we need only consider components up to P_6, since larger components could only result in combinations of components involving P_j, $j \geq 4$, which either should not participate in reactions (in the case of P_4) or should break-down into a P_4 and some smaller component. The programmed reaction matrix is

$$\widetilde{\mathbf{A}}_{\mathbf{q}} = \left(\begin{array}{lllll} -2+2q_{2,2} & -1+q_{3,2}+3q_{3,3} & -1 & 0 & -1+2q_{2,2} \\ q_{2,1} & -1+q_{3,2} & 0 & 0 & q_{2,1} \\ 0 & q_{3,1} & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right.$$

$$\left. \cdots \begin{array}{llll} 0 & 1 & 2q_{2,2} & 2q_{2,2} \\ -2 & -1 & -1+q_{2,1} & q_{2,1} \\ 0 & -1 & 0 & -2 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right),$$

and should be compared to the natural reaction matrix $\mathbf{A}$ for six parts shown in Section IV-A.

As an example, suppose that the rates $k_f(\mathbf{a})$ are 3, 3, 5, 4, 4, 3, 1, 5 and 4, taken in the same order as the actions appear in $\mathbf{A}$. The reverse rates are determined using a bond energy $\varepsilon = 1$. Starting with the macrostate $\mathbf{v}_0 = (6\ 0\ 0\ 0\ 0\ 0\ 0)$ gives 11 states. In this case, the optimal probabilities are

$$\mathbf{q} = (q_{2,1}, q_{2,2}, q_{3,1}, q_{3,2}, q_{3,3})^T = (\frac{1}{2}, \frac{1}{2}, 1, 0, 0)^T,$$

which gives an average yield of 0.72 components of type P_4 at equilibrium.

In this example, the obvious (and suboptimal) choice may seem to be $\mathbf{q} = (1\ \ 0\ \ 1\ \ 0\ \ 0)^T$, where one keeps components of type P_2 when they form. We call this the *greedy* choice. However, in this example (due to the choice of reaction rates), components of type P_3 form quickly and require an additional P_1 to become P_4s. Figure 5 shows the transient responses for the optimal and greedy assignment of probabilities.

Remark: This formulation depends on the number of parts assumed to be present initially in the system, since each choice for n results in a different $\mathbf{S}$ and therefore $\mathbf{Q}$. However, we have noticed empirically that: (1) the probabilities $\mathbf{q}$ resulting from different choices of n are often the same or nearly the same; and (2) when the probabilities found via optimization using different values for n are plugged in to the mass action kinetics equations (which assume a continuum of parts), the resulting equilibria are very close. We hope to report on these interesting phenomena more completely in a later paper.

B. The Programmable Parts Example

To illustrate the optimization method with the programmable parts, we consider the problem of assembling hexagons (C_{19} in Figure 3). The goal is to use the large set of rate data (Figure 4) intelligently to make the best hexagon-forming program possible.

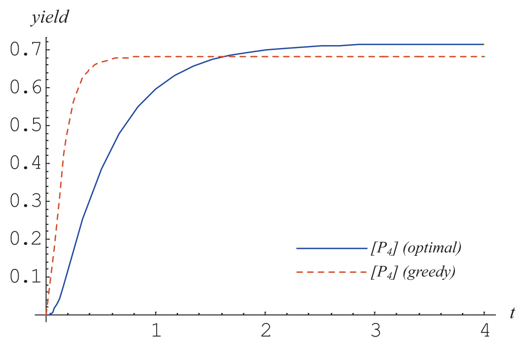


Fig. 5. The expected number of components of type P_4 as a function of time for the optimal and greedy choices of $\mathbf{q}$.

A hexagon is grown from components C_1, C_2, C_3, C_5 and C_{10}. One may also wish to allow "scaffold" assemblies like C_8 and hope they later react with, for example, a C_3 to form a component that contains a hexagon. There are hundreds of other component types that can result from interactions among this small set of component types and we have measured hundreds of non-zero reaction rates for these interactions. This results in a very large $\mathbf{Q}$ matrix. Furthermore, for each component type that does not contain a hexagon as a sub-component, one has a choice of how to break it down according to $\mathbf{q}$.

However, much can be gleaned from smaller subproblems. For example, we have noticed in experiments that C_{10} hardly ever reacts with other assemblies. The framework above allows us to determine the optimal way to handle the appearance of C_{10}: We consider how to break C_{10} into the following six options:

$$\begin{array}{ll} 1: C_{10} & 4: C_3 + 2C_1 \\ 2: C_5 + C_1 & 5: C_2 + 3C_1 \\ 3: C_3 + C_2 & 6: 5C_1, \end{array}$$

to which we associate the probabilities $q_{10,i}$, $i \in \{1, ..., 6\}$. To reduce the state space, we use the *greedy choice* for reactions that create sub-components of type C_{19} and the *reject choice* for all other reactions (meaning that we simply undo these reactions as soon as they happen). Starting with 9 parts and enumerating the states according to the procedure described in Section III-C results in 26 states and a 26×26 dimensional $\mathbf{Q}(\mathbf{q})$ matrix.

Running our optimization code on this problem using the measured reaction rates gives $q_{10,2} = 1$ and all other options equal to zero[1]. We also ran the optimization assuming other values for n, and obtained the same value for $\mathbf{q}$. For example, with 16 parts $\mathbf{Q}$ is 136×136 dimensional.

To test the approach, we compared the optimal versus the greedy choice using the mass action kinetics model of the dynamics, the high-fidelity simulation and the robot testbed. The results are shown in Figure 6. The data agree qualitatively, with the details in, for example, time scale differ due to the different assumptions on how communications are modeled compared to the somewhat slow communication rate in the actual robots.

[1]To ensure that $\mathbf{Q}(\mathbf{q})$ admits only one stationary distribution and is physically realistic [18, ch. 5], we have assumed that the reverse reactions that we did not observe in collecting data nevertheless have non-zero rate constants, which we have taken to be an order of magnitude slower than the slowest measured rate.

VII. A Universal Graph Grammar Implementation

To implement programmed assembly actions with robotic parts, we have designed a communications protocol that keeps each robot up-to-date about the component-type it is a part of and the role it takes in that component. The protocol is easily expressed as a *graph grammar* [13], which is a set Φ of rules of the form $L \rightharpoonup R$ where L and R are *simple labeled graphs*. If the some part of the system matches the graph L it can be replaced by the graph R in a distributed manner.

When an assembly event occurs, the topology of the robot network changes, and the robots use a *graph recognizer* [20], [21] protocol to determine the new topology. We have modified the notion of a graph recognizer to also include information concerning the *role* that each node takes in the graph.

Here we describe the protocol for the polymerization example. We briefly discuss the protocol for the programmable parts, which is similar, but considerably more tedious to describe.

A. Polymerization Protocol

At all times, each robot in the system has a *label* of the form (i, j, σ) where $i, j \in \mathbb{N}$ record the component type and role and $\sigma \in \Sigma$ is a symbol used to store temporary information. Initially, all robots are disconnected, and each has the label $(1, 1, .)$, meaning that each has role 1 in a component of type 1. The symbol "." is used to indicate a resting state. A chain of, say, four robots in a resting state has the form

$$(4, 1, .) - (4, 2, .) - (4, 3, .) - (4, 4, .).$$

We suppose that the protocol executes at a much higher rate than the natural forward and reverse rates with which components combine. In practice, we use time-stamps along with the labels described below to resolve conflicts wherein two updates are happening simultaneously.

(Formation) When two components P_i and P_k interact, they form P_{i+k}. This could happen in four different ways, depending on which ends interact. For example, suppose they bind via the robots labeled $(i, i, .)$ and $(k, k, .)$. The following rules manage the recognition of the new situation. In the first rule, we require that $i < k$ and in the second and third rules, we require that $i \neq k$.

$$\begin{array}{rcl} (i,i,.) \;\; (k,k,.) & \rightharpoonup & (i+k,k+1,g) - (i+k,k,s) \\ (i,j,s) - (k,l,.) & \rightharpoonup & (i,j,.) - (i,j-1,s) \\ (i,j,.) - (k,l,g) & \rightharpoonup & (k,l+1,g) - (k,l,.) \\ (k,k,g) & \rightharpoonup & (k,k,.). \end{array}$$

The interpretation of, for example, the first rule, is: If some robot labeled $(i, i, .)$ interacts with a robot labeled $(k, k, .)$, then they bind and change their labels to $(i+k, k+1, g)$ and $(i+k, k, s)$ respectively. The symbols g and the s initiate a

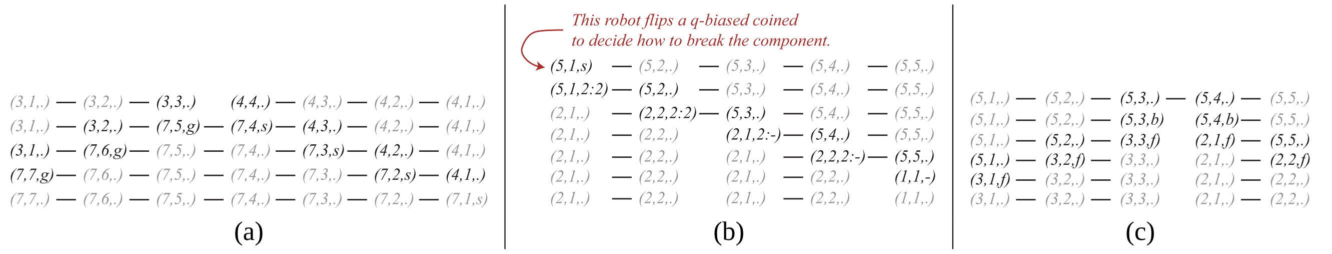

(3,1,.) — (3,2,.) — (3,3,.) (4,4,.) — (4,3,.) — (4,2,.) — (4,1,.)
(3,1,.) — (3,2,.) — (7,5,g) — (7,4,s) — (4,3,.) — (4,2,.) — (4,1,.)
(3,1,.) — (7,6,g) — (7,5,.) — (7,4,.) — (7,3,s) — (4,2,.) — (4,1,.)
(7,7,g) — (7,6,.) — (7,5,.) — (7,4,.) — (7,3,.) — (7,2,s) — (4,1,.)
(7,7,.) — (7,6,.) — (7,5,.) — (7,4,.) — (7,3,.) — (7,2,.) — (7,1,s)
(a)
This robot flips a q-biased coined to decide how to break the component.
(5,1,s) — (5,2,.) — (5,3,.) — (5,4,.) — (5,5,.)
(5,1,2:2) — (5,2,.) — (5,3,.) — (5,4,.) — (5,5,.)
(2,1,.) — (2,2,2:2) — (5,3,.) — (5,4,.) — (5,5,.)
(2,1,.) — (2,2,.) (2,1,2:-) — (5,4,.) — (5,5,.)
(2,1,.) — (2,2,.) (2,1,.) — (2,2,2:-) — (5,5,.)
(2,1,.) — (2,2,.) (2,1,.) — (2,2,.) (1,1,-)
(2,1,.) — (2,2,.) (2,1,.) — (2,2,.) (1,1,.)
(b)
(5,1,.) — (5,2,.) — (5,3,.) — (5,4,.) — (5,5,.)
(5,1,.) — (5,2,.) — (5,3,b) (5,4,b) — (5,5,.)
(5,1,.) — (5,2,.) — (3,3,f) (2,1,f) — (5,5,.)
(5,1,.) — (3,2,f) — (3,3,.) (2,1,.) — (2,2,f)
(3,1,f) — (3,2,.) — (3,3,.) (2,1,.) — (2,2,.)
(3,1,.) — (3,2,.) — (3,3,.) (2,1,.) — (2,2,.)
(c)

Fig. 7. Examples of the universal grammar in action. (a) Two chains bind. The *formation rules* update the labels of the robots to reflect the size of the new component and their roles in it. (b) A newly formed P_5 dismantles itself using the *destruction rules* into $2P_2 + P_1$. (c) A P_5 is broken by some external force and the robots adjust their labels to reflect the new situation using the *uncontrolled break rules*.

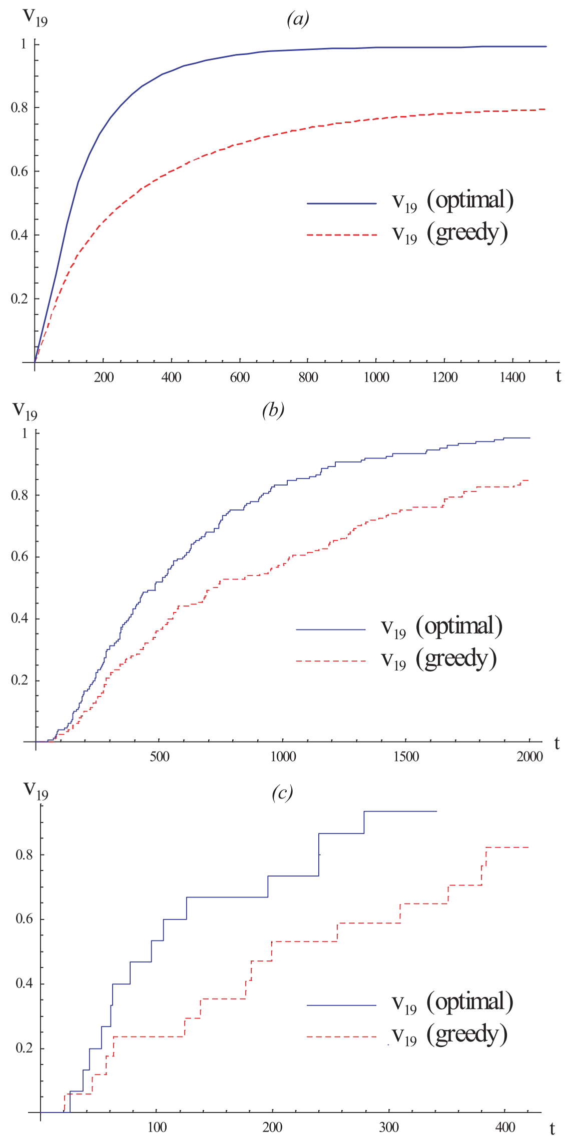


Fig. 6. Comparison of the optimal choice and the greedy choice for how to dismantle C_{10} when assembling C_{19}. (a) Mass action kinetics with an initial concentration of nine parts per square meter. (b) The average of 256 trajectories from a high-fidelity mechanics-based simulation of the robots starting with nine parts. (c) Data averaged over 15 runs of our programmable parts testbed with nine parts and the universal grammar. The data was collected using an overhead camera and subsequent software analysis of the video data. The different time scales are likely due to different communication rates: immediate in (a), fast in (b) and somewhat slow in (c), the actual experiment.

cascade of rule applications to the rest of the robots in the chain, as illustrated in Figure 7(a). The rules describing the update resulting from the chains combining via $(i, 1, .)$ and $(k, k, .)$ etc., are similar.

(Destruction) When a chain is first formed, it must be dismantled into smaller, more useful components according to the probabilities $q_{i,j}$ determined by the optimization described in Section VI. We appoint the node labeled $(k, 1, s)$ to make this choice by using a random number generator. If the node chooses to break the chain into parts of size $p_1, ..., p_r$, then it throws a rule of the form

$$(k, 1, s) \rightharpoonup (k, 1, h : t)$$

where $h = p_1$ and $t = \{p_2, ..., p_{r-1}\}$ is a *head-tail* description of the list of part sizes (as used in functional programming, for example). We then have the following rules.

$$\begin{array}{rcl}(i, j, h : t) - (k, l, .) & \rightharpoonup & (h, j, .) - (h, j + 1, h : t) \quad h \neq j \\ (i, h, h : t) - (k, l, .) & \rightharpoonup & (h, h, .) \;\; (hd(t), 1, hd(t) : rest(t)) \\ (h, h, h : \varnothing) & \rightharpoonup & (h, h, .).\end{array}$$

Notice that the second rule deletes an edge (or breaks a bond) in a controlled way to execute the required dismantling. These rules are illustrated in Figure 7(b).

(Uncontrolled Breaks) When a reverse reaction occurs naturally, a chain is broken into two smaller chains. A graph grammar rule that models this is

$$(i, j, .) - (i, j + 1, .) \;\; \rightharpoonup \;\; (i, j, b) \;\; (i, j + 1, b)$$

where the symbol b denotes the fact that the part has just broken a bond. This rule is not part of the communications protocol. Rather, the robots can sense when their neighbors disappear, and they set the third component of their labels to "b" accordingly. We then use one other symbol, "f", to mean "fixed" in the following rules to *recover* from the break. The first three repair the states of the lower half of a broken chain.

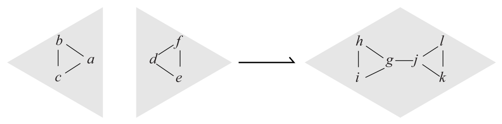
$$\begin{array}{rcl}(i, j - 1.) - (i, j, b) & \rightharpoonup & (i, j - 1, .) - (j, j, f) \\ (i, j - 1, .) - (k, j, f) & \rightharpoonup & (k, j - 1, j) - (k, j, .) \\ (k, 1, f) & \rightharpoonup & (k, 1, .).\end{array}$$

The second three repair the upper half.

$$\begin{array}{rcl}(i, j, b) - (i, j + 1, .) & \rightharpoonup & (i - j + 1, 1, f) - (i, j + 1, .) \\ (i, j, f) - (k, l, .) & \rightharpoonup & (i, j, .) - (i, j + 1, f) \\ (i, i, f) & \rightharpoonup & (i, i, .).\end{array}$$

B. Programmable Parts Protocol

The programmable parts use a very similar protocol to implement programmed reactions, which we discuss briefly. The two main complications are: (1) The programmable parts can form components more complicated than paths; and (2) the role of each face of each part must be identified. The first complication can be handled by maintaining a spanning tree of each component as components form and adapting the rules in the previous subsection to trees. The second complication requires the use of rules of the form



where $a \dots l$ are compound labels such as those used in the previous subsection. The grey triangles indicate how the triangular labeled graphs are embedded in the plane. Rule sets for programmable parts are still graph grammars. However, the orientation of the triangles must be maintained in rule application, slightly complicating the formalism, as described elsewhere [13].

VIII. Discussion

The main contributions of this paper are (1) careful modeling of programmed stochastic self-assembly in the language of kinetics; (2) the recognition that, in our systems, local interaction rules simply accept, reject or redirect binding events initiated by the environment; (3) a formulation of the problem that allows us to find interaction rules that result in optimum performance; and (4) the application of these ideas to a variety of examples, including our experimental testbed for which we have meticulously measured natural kinetic rate constants as input into the optimization problem.

We believe that our notion of reaction pathway tuning can be applied to a variety of systems (we are investigating directed self-assembly at other scales in our lab) and using a variety of objective functions that not only optimize yield, but also the overall rate, the strength of a desired pathway for a particular component, the probability of oscillations, robustness to model-uncertainty, etc. In the future, we plan to explore and extend these ideas.

The main drawback of the approach is that the complexity of the optimization problem, while polynomial in the size of $\mathbf{Q}$, is exponential in the number of component types. Nevertheless, for the issues we encounter with the programmable parts testbed, solving small problems is quite insightful. In the future, we plan to improve our implementation of the optimization problem (which presently is not nearly as efficient as it could be) to handle much larger instances and explore approximation methods that give, perhaps, suboptimal solutions, but do so quickly.

Acknowledgments

This work is partially supported by NSF Grant #0347955: *CAREER: Programmed Robotic Self-Assembly*. S. Burden is also partially supported by a *Mary Gates Undergraduate Research Training Fellowship*. We are indebted to Mehran Mesbahi for his helpful suggestions regarding solving the optimization problem in Section VI.

References

[1] E. Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure and Dynamics*, 11(2):263–270, May 2000.

[2] R. C. Mucic, J. J. Storhoff, C. A. Mirkin, and R. L. Letsinger. DNA-directed synthesis of binary nanoparticle network materials. *Journal of the American Chemical Society*, 120:12674–12675, 1998.

[3] N. Bowden, A. Terfort, J. Carbeck, and G. M. Whitesides. Self-assembly of mesoscale objects into ordered two-dimensional arrays. *Science*, 276(11):233–235, April 1997.

[4] A. Greiner, J. Lienemann, J. G. Korvink, X. Xiong, Y. Hanein, and K. F. Böhringer. Capillary forces in micro-fluidic self-assembly. In *Fifth International Conference on Modeling and Simulation of Microsystems (MSM'02)*, pages 22–25, April 2002.

[5] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Self-organizing programmable parts. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ Robotics and Automation Society, 2005.

[6] Nils Napp, Sam Burden, and Eric Klavins. The statistical dynamics of programmed robotic self-assembly. In *International Conference on Robotics and Automation*, 2006. Submitted.

[7] P. White, V. Zykov, J. Bongard, and H. Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics Science and Systems*, Boston, MA, June 2005.

[8] H. Alper, C. Fischer, E. Nevoigt, and G. Stephanopoulos. Tuning genetic control through promoter engineering. *PNAS*, 102(36):12678–12683, 2005.

[9] K. Hosokawa, I. Shimoyama, and H. Miura. Dynamics of self-assembling systems: Analogy with chemical kinetics. *Artifical Life*, 1(4):413–427, 1994.

[10] P. J. White, K. Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of the International Conference on Robotics and Automation*, New Orleans, LA, 2004.

[11] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit module. *Autonomous Robots*, 10(1):107–124, January 2001.

[12] J. Suh, S. Homans, and M. Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *IEEE International Conference on Robotics and Automation*, pages 4095–4101, Washington, D.C., May 2002.

[13] E. Klavins, R. Ghrist, and D. Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962, June 2006.

[14] K. Saitou. Conformational switching in self-assembling mechanical systems. *IEEE Transactions on Robotics and Automation*, 15(3):510–520, 1999.

[15] M. Fienberg. The existence and uniqueness of steady states for a class of chemical reaction networks. *Archive for Rational Mechanics and Analysis*, 132:311–370, 1995.

[16] R. M. Dirks, M. Lin, E. Winfree, and N.A. Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Research*, 32(4):1392–1403, 2004.

[17] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.

[18] D.W. Strook. *An Introduction to Markov Processes*. Springer-Verlag, 2005.

[19] PENOPT. http://www.penopt.com/.

[20] I. Litovsky, Y. Métevier, and W. Zielonka. The power and limitations of local computations on graphs and networks. In *Graph-theoretic Concepts in Computer Science*, volume 657 of *Lecture Notes in Computer Science*, pages 333–345. Springer-Verlag, 1992.

[21] B. Courcelle and Y. Métivier. Coverings and minors: Application to local computations in graphs. *European Journal of Combinatorics*, 15:127–138, 1994.