Directed Self-Assembly Using Graph Grammars

Eric Klavins*

University of Washington
Electrical Engineering Department
Box 352500
Seattle, WA 98195

Abstract. In this paper we describe the graph grammar approach to modeling self-assembly. The approach is used to describe how the *topology* of an assembling aggregate changes as it grows. The main purpose of the paper is to demonstrate the utility of the approach by giving detailed examples. We also describe the beginnings of our approach to physically embedding graph grammar assembly rules in physical settings, focusing on macro- and micro-scale programmable parts and a simulation environment.

1 Introduction

Engineering in the realm of the very small presents us with the daunting problem of manipulating and coordinating vast numbers of objects so that they perform some global task. Nevertheless, there are examples of sophisticated machines, such as the ribosome or the mechanical motor in the bacterial flagellum, that seem to be built *in bulk* spontaneously. It is assumed that in the formation of these objects, simple small components *self-assemble* into more complex aggregates which, in turn, self-assemble into larger aggregates.

Our starting point in understanding self assembly is the idea of *conformational switching* [17]: Each part (molecule, robot, etc.) exists in one of several *conformations* or shapes. When two parts come into close proximity, they attach or not based on whether their conformations are complimentary. If they do attach, their conformations change (mechanically for example), thereby determining in what future assembly interactions the parts may partake.

In this paper, as in other work [16, 14], we represent the conformation of a part by a discrete symbol, and we model an assembly as a simple graph labeled by such symbols. Vertices in these graphs represent parts, and the presence of an edge between two parts represents that they are somehow attached. An *assembly rule*, then, is a pair of such labeled graphs interpreted as follows. If a subset of parts with their labels and edges matches the first part of an assembly rule, then it may be replaced by the second part of the rule to achieve a new state of the system. We are in particular interested in (1) the situation where the parts decide in a distributed fashion whether and how to execute an assembly rule;

* This work is supported in part by NSF Grant #0347955.

(2) how to model natural and artificial assembly processes; (3) how to define assembly rule sets so that only prespecified assemblies are built.

Specifically, the main contribution of this paper is a set of illustrative examples that show (1) how various types of assembly processes can be modeled and (2) how the graph grammars can simulate string grammars [9] and tile assembly systems [21, 2]. The paper is meant to be complimentary to another paper where we investigate some of the formal properties of graph grammars [14]. We also include a brief discussion of how one can embed the graph grammar formalism into physical assembly systems, including macro-scale robotic "programmable parts" and micro-scale directed tile assembly. Finally, we end with a brief discussion of related work.

2 Definitions

We begin with basic definitions. Much of this section appeared first elsewhere [14], we include it for completeness. Basic graph-theory definitions are not numbered, but only recalled [6].

A *simple labeled graph* over an alphabet Σ is a triple $G = (V, E, l)$ where V is a set of *vertices*, E is a set of pairs or *edges* from V, and $l : V \to \Sigma$ is a labeling function. We restrict our discussion to simple labeled graphs and thus simply use the term *graph*. We denote by V_G, E_G and l_G the vertex set, edge set and labeling function of the graph G or by V, E and l when there is no danger of confusion. We will usually use the alphabet $\Sigma = \{a, b, c, ...\}$.

Given graphs G_1 and G_2, we write $f : G_1 \to G_2$ and $f : V_{G_1} \to V_{G_2}$ equivalently to mean that f is a function from the vertex set of G_1 to the vertex set of G_2. A function $h : G_1 \to G_2$ is a label preserving *embedding* if

1. h is injective,
2. $\{x, y\} \in E_{G_1} \Leftrightarrow \{h(x), h(y)\} \in E_{G_2}$,
3. $l_{G_1} = l_{G_2} \circ h$.

If h is also surjective then it is called an *isomorphism*. The graphs G_1 and G_2 are said to be *isomorphic* (written $G_1 \simeq G_2$) if there exists an isomorphism relating them.

Definition 1. *A* rule *is a pair of graphs $r = (L, R)$ where $V_L = V_R$. The graphs L and R are called the* left hand side *and* right hand side *of r respectively. The* **size** *of r is $|V_L| = |V_R|$. Rules whose vertex sets have one, two and three vertices are called* unary, binary *and* ternary, *respectively.*

We may refer to rules as being *constructive* ($E_L \subset E_R$), *destructive* ($E_L \supset E_R$) or *mixed* (neither constructive or destructive). A rule is *acyclic* if its right hand side contain no cycles (the left hand side may contain cycles).

Definition 2. *A rule r is* applicable *to a graph G if there exists an embedding $h : L \to G$. In this case the function h is called a* witness. *An* action *on a graph G is a pair (r, h) such that r is applicable to G with witness h.*

Definition 3. *Given a graph $G = (V, E, l)$ and an action (r, h) on G with $r = (L, R)$, the* application *of (r, h) to G yields a new graph $G' = (V', E', l')$ defined by*

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

We write $G \xrightarrow{r,h} G'$ to denote that G' was obtained from G by the application of (r, h).

Definition 4. *A* graph assembly system *is a pair (G_0, Φ) where G_0 is the* initial graph *and Φ is a set of rules (called the* rule set*).*

We often refer to a system simply by its rule set Φ and assume that the initial graph is the infinite graph defined by

$$G_0 \triangleq (\mathbb{N}, \varnothing, \lambda x.a) \tag{1}$$

where $a \in \Sigma$ is the *initial symbol* (here $\lambda x.a$ is the function assigning the label a to all vertices).

Definition 5. *An* assembly sequence *of a system (G_0, Φ) is a finite sequence $\{G_i\}_{i=0}^k$ such that there exists a sequence of actions $\{(r_i, h_i)\}_{i=1}^k$ where $r_i \in \Phi$ and*

$$G_i \xrightarrow{r_i,h_i} G_{i+1}$$

for $i \in \{0, ..., k-1\}$.

Thus, a system (G_0, Φ) defines a non-deterministic dynamical system whose states are labeled graphs over V_{G_0}. The system is non-deterministic since, at any step, many rules in Φ may be simultaneously applicable, each possibly via several witnesses.

Two vertices in a graph G are *connected* if there is a path (sequence of edges) connecting them in G. The *connectivity* relation on V is an equivalence relation partitioning V into sets $\{V_i\}_{i \in I}$ where v_1 and v_2 are connected if and only if $v_1, v_2 \in V_i$ for some i. The sets V_i are called the *components* of G. A graph G is connected if it has exactly one component.

Definition 6. *A connected graph G is* reachable *in a system (G_0, Φ) if there exists an assembly sequence $\{G_i\}_{i=0}^k$ of (G_0, Φ) such that G is isomorphic to some component of G_k. The set of all such reachable graphs is denoted $\mathcal{R}(G_0, \Phi)$, or just $\mathcal{R}(\Phi)$ if G_0 defined by (1).*

Definition 7. *A graph $G \in \mathcal{R}(G_0, \Phi)$ is* stable *if for all G' there does not exist an action (r, h) on the disjoint union $G \amalg G'$ such that $r = (L, R) \in \Phi$ and $h(L) \cap V_G$ is nonempty. The set of all such stable graphs is denoted $\mathcal{S}(G_0, \Phi)$, or just $\mathcal{S}(\Phi)$ if G_0 defined by (1).*

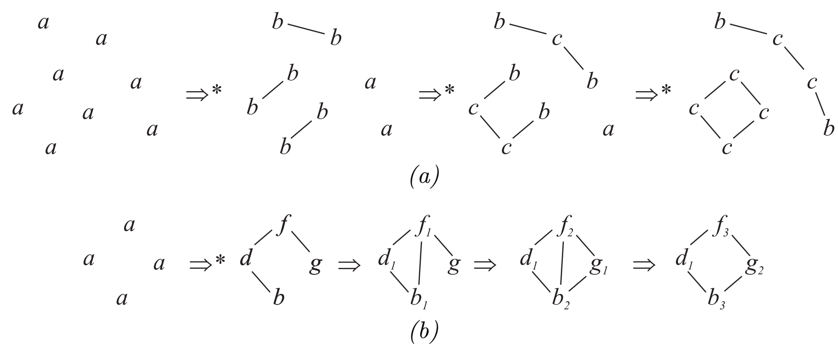


Fig. 1. Example assembly sequences arising from the rule sets defined in Examples 1 and 2. The symbol $\Rightarrow^*$ denotes the application of more than one rule and, in the first transition of *(a)*, for example, demonstrates the concurrent application of rules. *(a)* Φ_1 produces unstable chains and stable cycles. *(b)* The stable set of Φ_2 is exactly the four-cycle.

3 Examples

In this section we describe several examples. Our goal is to illustrate the utility of the graph grammar approach to defining or characterizing self assembling systems. Thus, we attempt with these examples to span a wide spectrum of phenomena.

We begin by illustrating the approach with a simple example.

Example 1. Define a constructive rule set by

$$\Phi_1 = \begin{cases} a \quad a \quad \Rightarrow \quad b - b, \\ a \quad b \quad \Rightarrow \quad b - c, \\ b \quad b \quad \Rightarrow \quad c - c \end{cases}$$

We use the position of the nodes in the presentation of the rules to denote the re-labeling. For example, the first rule in Φ_1 is given by

$$L = (\{1, 2\}, \varnothing, \lambda x.a)$$
$$R = (\{1, 2\}, \{\{1, 2\}\}, \lambda x.b)$$

An example assembly sequence for Φ_1 set is shown in Figure 1(a). The reachable set $\mathcal{R}(\Phi_1)$ consists of all cycles and chains. Only cycles are stable however. This is because, once a chain closes into a cycle (using the last rule), all of its nodes are labeled by c, which does not appear in the left hand side of any rule. Thus, the stable set $\mathcal{S}(\Phi_1)$ consists only of cycles. ▲

The rules in example 1 are not useful if cycles of a particular size are desired. In fact, it can be shown [14] that if a set of binary rules rules produces a cyclic graph of size n, then it produces cyclic graphs of size kn for all $k \in \mathbb{N}$ (the result is actually somewhat more general). The next example shows how get around this using larger rules.

Example 2. Define a mixed rule set with three binary constructive rules, two ternary constructive rules and one binary destructive rule by

$$\Phi_2 = \begin{cases} a \quad a \;\Rightarrow\; b - c, & \begin{array}{ccc} & d & \\ \diagup & & \diagdown \\ b & & f \end{array} \;\Rightarrow\; \begin{array}{ccc} & d_1 & \\ \diagup & & \diagdown \\ b_1 & \text{---} & f_1 \end{array}, \\ a \quad c \;\Rightarrow\; e - d, & \begin{array}{ccc} & f_1 & \\ \diagup & & \diagdown \\ b_1 & & g \end{array} \;\Rightarrow\; \begin{array}{ccc} & f_2 & \\ \diagup & & \diagdown \\ b_2 & \text{---} & g_1 \end{array}, \\ a \quad e \;\Rightarrow\; g - f, & b_2 - f_2 \;\Rightarrow\; b_3 \quad f_3 \end{cases}$$

An example assembly sequence for Φ_2 is shown in Figure 1(b). The three binary constructive rules yield chains of length 4. The two constructive and cyclic ternary rules "triangulate" the cycle. The last rule, removes the first triangulating edge to yield a length 4 cycle, which is the unique stable graph of the system. ▲

Graph grammars are a generalization of grammars on strings studied in computer science. Thus, they can do anything that string grammars can do (represent regular languages, arbitrary computation, ...). In the next example, we show how to simulate a string grammar in *Chomsky Normal Form* [9] with a graph grammar. From the example, it should be clear that any string grammar, context-free or context sensitive, can be simulated in a similar fashion.

Example 3. A *context free* string grammar is said to be in *Chomsky Normal Form* if all of its productions are of the form $A \to BC$ or $A \to x$ where B and C are not the start symbol S and x is a terminal symbol. To simulate a string grammar Γ with a graph grammar system (G_0, Φ), we use the alphabet

$$\Sigma_\Gamma \cup \{\tau\}$$

where Σ_Γ consists of all terminal and non-terminal symbols from Γ and τ is a new symbol. To represent the production $A \to BC$, we use the set of rules of the form

$$\begin{array}{ccc} & \tau & \\ *_1 & \text{———} & A \text{ —— } *_2 \end{array} \quad \Rightarrow \quad \begin{array}{ccc} & B & \\ & \diagup \quad \diagdown & \\ *_1 & & C \text{ —— } *_2 \end{array}$$

where $*_1$ and $*_2$ range over all symbols in Σ or may be left out to include situations wherein A is at the end of the current derivation. Similarly, for the production $A \to x$, we use a set of rules of the form

$$*_1 \text{ —— } A \text{ —— } *_2 \quad \Rightarrow \quad *_1 \text{ —— } x \text{ —— } *_2$$

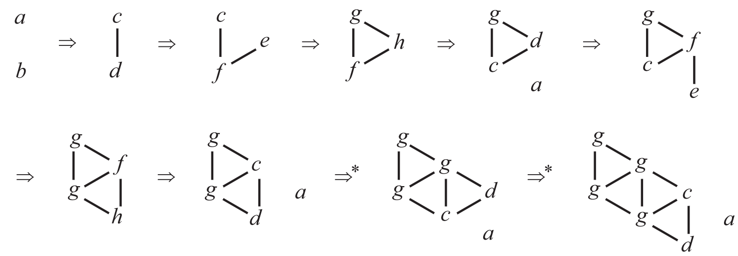


Fig. 2. An assembly sequence arising from the rule set defined in Example 4 that produces arbitrarily long "trusses". Although there are an infinite number of nodes labeled a, we only show them if they are about to be used in a rule application. Note that the sub-graph $c - -d$ appears at the beginning of each new truss section.

where, once again, $*_1$ and $*_2$ range over all symbols in Σ or may be left out. The initial graph G_0 contains an infinite discrete graph all of whose vertices are labeled τ except one, which is labeled with S, the start symbol of Γ. The reachable set of the graph grammar consists of all chains of terminal and non-terminal symbols corresponding to derivations in Γ, and the stable set consists of chains of terminal symbols corresponding to strings in the language $L(\Gamma)$ generated by G. ▲

We believe the utility of the graph grammar approach is not in its ability to model computation, but in representing the topology of assemblies. In the next example, we illustrate how repeating truss-like structures can be formed using a simple set of rules.

Example 4. Define a rule set by

$$\Phi_4 = \begin{cases} a \quad b \;\Rightarrow\; c - d, \\ a \quad d \;\Rightarrow\; e - f, \\ c \quad e \;\Rightarrow\; g - h, \\ f - h \;\Rightarrow\; c - d \end{cases}$$

and define an initial graph to be discrete, with each node labeled a except one "seed" node labeled b. The reachable set of this grammar consists of truss-like structures. There are no stable trusses, as there is always a way to extend a truss of a given length. Figure 2 illustrates an assembly sequence for this grammar. ▲

Of course, more sophisticated grammars can be defined to produce even more complicated structures.

A crucial problem for self assembly is: *Given a desired graph G_d, produce a set of assembly rules Φ so that $\mathcal{S}(\Phi) = \{G\}$ and $G \simeq G_d$.* The solution to this problem could be used as the basis for an automatic self-assembly design tool, and turns out to be solvable, as the next example, which assumes that G_d is acyclic, suggests.

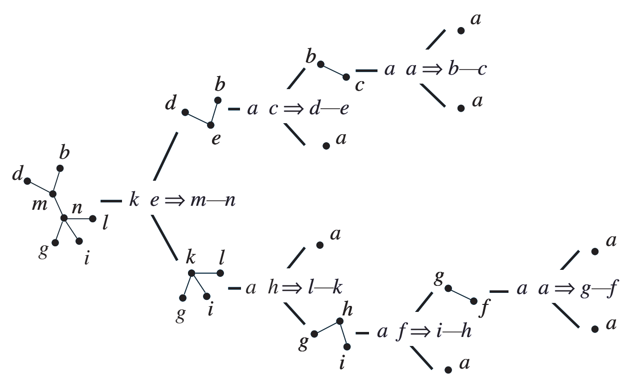


Fig. 3. A rule set that produces a given tree structure as uniquely stable can be automatically obtained by disassembling the tree and defining rules to build it from the bottom up.

Example 5. Given an acyclic graph $T = (V, E)$, we can define a rule set Φ_T such that T is uniquely stable with respect to the system (G_0, Φ_T). We proceed by first disassembling T into a *binary assembly graph*, as illustrated in Figure 3. The root of the assembly graph is T, inner nodes are sub-trees of T, leaf nodes are singletons. The children of a sub-tree are obtained by deleting an edge from the subtree. Rules are then defined that reassemble the tree from the bottom up, as shown in Figure 3. Two new labels are used in each rule. It can be shown that the resulting rule set, which contains $|V|-1$ rules and uses $2|V|-2$ labels, makes (a graph isomorphic to) the tree T as uniquely stable. ▲

In fact, a rules set can be automatically synthesized to make *any desired graph* uniquely stable using only binary and ternary rules [14]. The procedure is to use the above algorithm to grow a spanning tree of the desired graph, and then triangulate the cycles in the graph using a procedure similar to that in Example 2. In another paper [14], we explain that this triangulation procedure using ternary rules is necessary: *There does not exist a binary rule set that produces a uniquely stable cyclic graph.*

A graph grammar need not define an assembly process wherein an assembly is "grown" from smaller parts. Non-stationary dynamical systems can be defined too, as the next example illustrates.

Example 6. A rule set need not define an assembly process, as in the following set, containing constructive and destructive rules as well as mixed rules that simply re-label nodes.

$$\Phi_6 = \begin{cases} e \quad g & \Rightarrow \quad c - h, \\ a - c & \Rightarrow \quad d \quad e, \\ b - h & \Rightarrow \quad f - b, \\ d - f & \Rightarrow \quad g - a \end{cases}$$

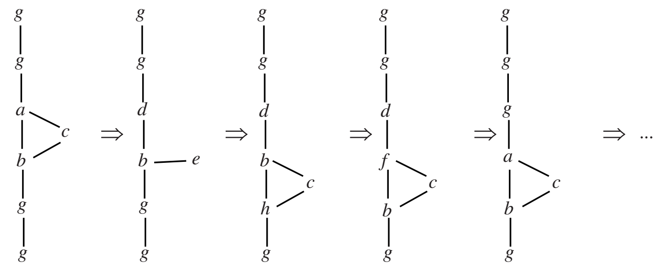


Fig. 4. The rule set defined in example 6 defines a dynamical system, wherein a part "ratchets" along a substrate.

An assembly sequence of Φ_3 is shown in Figure 4. The sequence starts with a cycle labeled with a, b and c attached to linear substrate of parts labeled g. As the figure shows, the rule set "ratchets" the 3-cycle along the substrate. ▲

There are other models of self-assembly besides graph grammars, as discussed in Section 5. For example, many researchers consider passive *tile systems* wherein the parts to be assembled do not have control over their states. Instead, 2D polygonal tiles combine along complimentary edges [1, 21, 2, 15]. What defines complimentary depends on the physical setting. For example, edges might consist of a strand of un-hybridized DNA: A complimentary strand on another tile will attach to it under suitable conditions [21].

A tile system is wedded to a particular geometry that cannot be modeled explicitly in graph grammars. However, the topology induced by the geometry can be modeled. For example, square tiles assembling on a planar surface produce a regular square lattice of tiles: Each tile is attached to four others. The next example shows how to simulate this situation. First we define a tile system in a similar manner to the tile systems defined elsewhere[21].

A (square) tile system (T, Σ, τ, R) consists of a discrete set T of tiles, an assignment $\tau : T \times \{N, S, E, W\} \to \Sigma$ of symbols to the north, south, east and west edges of each tile, and a symmetric relation $R \subseteq \Sigma \times \Sigma$ that specifies which pairs of edge symbols are complimentary.[1]The east edge of a tile may assemble with the west edge of another tile if the symbols on those edges are complimentary, and similarly for north and south. One usually starts with a seed tile $s \in T$ placed on a planar grid, and adds tiles one at a time following the rules above.

Example 7. To model a tile system (T, Σ, τ, R) with a graph grammar (Φ, G_0), we first define a new set of symbols. For each symbol $a \in \Sigma$, we add the "symbols"

$$(N, a),\ (S, a),\ (E, a),\ (W, a)\ \text{ and } (N, a)',\ (S, a)',\ (E, a)',\ (W, a)'$$

[1] The tile assembly model usually includes a binding energy for each pair of symbols, which we leave out for simplicity.

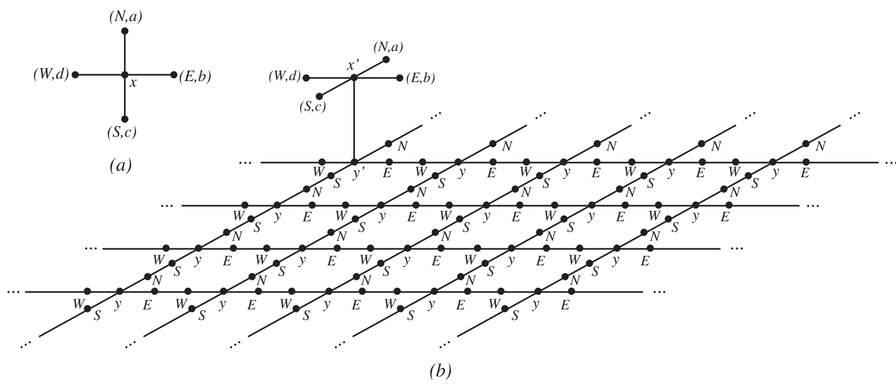


Fig. 5. *(a)* A graph representing a square tile with edges labeled a, b, c and d. *(b)* A graph representing the underlying grid or substrate onto which the tiles assemble.

to stand for the edges that tiles may have. The unprimed version of each symbol represents an unmatched edge and the primed version represents a matched edge. We also add the four new symbols

$$x,\ y,\ ,x',\ \text{and}\ y'.$$

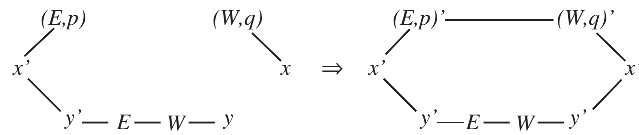
The symbols x and x' represent the "center" of a tile in different states: either attached to the underlying grid (primed) or not (unprimed). The symbols y and y' represent points on an underlying grid that are either unoccupied or not, respectively. Finally, we also add new symbols

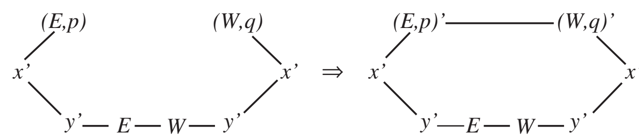
$$N,\ S,\ E,\ \text{and}\ W$$

that will be used to specify the orientation of the underlying grid.

The initial graph G_0 consists of two parts: the underlying grid and an infinite supply of tiles of each type. Figure 5(a) shows a graph that represents an unattached tile $t \in T$ where $\tau(t, N) = a$, $\tau(t, E) = b$, $\tau(t, S) = c$ and $\tau(t, W) = d$. The underlying grid, shown in Figure 5(b), initially consists of an infinite square lattice of nodes labeled by y separated by length 2 chains labeled with N and S or E and W. We also start with a copy of the seed tile attached to one of the y nodes.

To assemble tiles, it is not enough to simply combine tiles along complimentary edges. This could result in non-planar assemblies that are not possible in the tile assembly model. We must include information about how the tiles attach to the substrate. In effect, the topology of the initial underlying grid specifies the possible topologies of the resulting assembly (and could in fact be cylindrical, toroidal, etc). To mimic the tile assembly model, therefore, we add, for each $(p, q) \in R$ the two rules





and similar rules for north-south pairs. The first rule specifies how an unattached tile may bind to a tile already attached to the substrate via the east and west edges of the two tiles. The second rule specifies how two tiles already attached to the substrate but not to each other may bind. Note that without the E and W nodes in the substrate, the "edges" of the tiles could permute and direction would not be preserved. ▲

4 Physical Instantiations

4.1 A Particle Based Model

In our first physical embedding of the graph grammar formalism, we suppose that a large number of robotic parts *float* in a stirred fluid (Figure 6). Upon colliding (by chance), two parts will latch onto each other or not based on whether their current labels match the left hand side of a rule in a rules set Φ. If they do latch together, then they change their labels according to the rule. A similar scheme works for mixed or destructive rules.

To model this system, we suppose that each part i with position $x_i \in \mathbb{R}^3$ has a latch variable $L_{i,j} \in \{0,1\}$ associated with every other robot j. When the parts come in close proximity, they sense each other's current label in an attempt to find an applicable rule. If one is found, they both set their latch variable for the other to 1 and change their labels. Otherwise the latch variable is set to 0 and the parts bounce off each other. The dynamics of robot i are

$$m\ddot{x} = F_1(x_i, t) - c\dot{x}_i + \sum_{j \neq i} L_{i,j} L_{j,i} F_2(x_i, x_j) \qquad (2)$$

where F_1 is a time varying force field that models the effect of the fluid on the part and

$$F_2(x_i, x_j) \triangleq -\nabla_{x_i} U(x_i, x_j) - b \frac{\dot{x}_i (x_j - x_i)}{||x_j - x_i||} (x_j - x_i)$$

is a damped spring, with spring potential U, modeling the latching mechanism. We suppose that U has a minimum at $||x_i - x_j|| = R$, the desired distance between assembled particles.

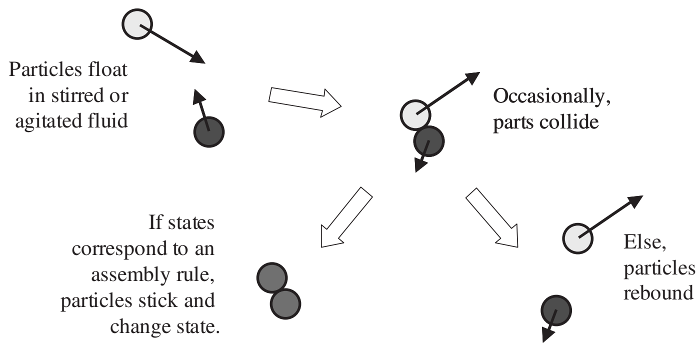


Fig. 6. A particle based embedding of self assembly using graph grammars.

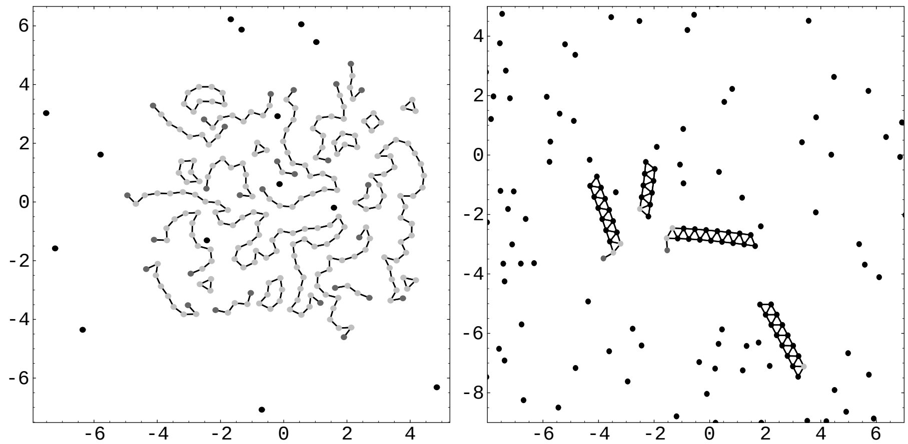
4.2 Programmable Parts with Modulated Surfaces

In our lab we are presently building simple macro-scale devices that can implement the scheme described above. The devices, which we call *programmable parts*, consist of tiles with magnetic latches on their edges. The parts float on an air table and are stirred by overhead fans. When two parts collide, they stick (because the default positions of their magnets are complementary). After they stick, they communicate their states to each other via IR transceivers mounted on their edges. If the states do not correspond to a rule, the parts switch their latches and repel away from each other. We will report on this effort in a future publication.

In other work [13] we describe, for example, how such a mechanism would work using capillary forces. In a related effort, we are in the preliminary stages of considering MEMs-scale tiles whose wetabilities can be modulated [5] to realize a controlled binding interaction.

4.3 Simulation Environment

We have begun to explore the behavior of the above model in simulation using the rule systems we explored in this paper and Equation (2). The simulation can handle thousands of parts by carefully considering the collision dynamics of only neighboring parts and by using simplifying assumptions on the effect of the fluidic stirring due to F_1 in Equation (2). In Figure 7*(a)*, we show a snapshot of a system of parts that use the rules in Example 1. In Figure 7*(b)*, we show a snapshot of a system using the truss-rules in Example 4. In our lab we are using the simulation environment to explore, for example, how quickly various rule systems assemble structures and how various design options affect other aspects of the assembly process. We are presently encoding the physics of more realistic systems with non-trivial geometries and binding site physics and plan to report on this effort in future work.



(a) (b)

Fig. 7. Simulation data using Equation (2) and *(a)* the rules from Example 1 and *(b)* the rules from Example 4.

5 Previous and Related Work

Conformational switching was first described as a symbolic process for self assembly by Saitou [16], who considered the assembly of strings in one dimension. Self assembly as a graph process has been described by the author of this paper [11, 12] (although the graph grammar formalism first appeared in [14]) and a rule synthesis procedure for trees was given that is somewhat more complex than the one described here. A method for using potential fields and deadlock avoidance to implement graph grammar rules with a group of mobile robots was also described [12].

Graph grammars were introduced [8, 7] at least two decades ago and have been used to describe a broad array of systems, from data structure maintenance to mechanical system synthesis. Graph grammars are, of course, a generalization of the standard "linear" grammars used in automata theory and linguistics [9] and thus (incidentally), can perform arbitrary computation. The use of graph grammars to model distributed assembly, to the best of our knowledge, is new.

There are other models of self assembly besides graph grammars, a complete list of which is beyond the scope of this paper. But, for example, several groups [21, 2, 3] have explored self assembly using passive *tiles* floating in liquid. The tiles attach along complimentary edges (due, for example, to capillary forces or the assembly of complimentary strands of DNA) upon random collisions. As illustrated in Example 7 above, graph grammars are somewhat more general and are possibly better suited to describing the *topology* of assemblies.

A simple dynamical model of the *physics* of tile assembly has been described [13]. Somewhat similar to the *stable set* in this paper (Definition 7), the identification of "unique" assemblies has been explored [18]. There is also other work on tile systems with conformal-like state information [10]. Such systems can in fact be used to perform arbitrary computations [20] and are best understood as two or three dimensional symbolic processes. Another approach uses geometrical constraints on part-part interactions to model, for example, the assembly of proteins into spherical shells called *capsids* [4]. The addition of simple processing to each part, similar in capability to that assumed in the present paper, is considered in models of the assembly of the T4 bacteriophage [19].

6 Conclusion

In this paper we have demonstrated that graph grammars can be used to model a variety of self-assembling systems and other systems consisting if local-rule-based interactions as well (such as the ratchet). The emphasis in graph grammars is on the algorithmic and topological structure of assembly as opposed to the geometrical or physical. This allows more generality than other models, which may be wedded to a particular geometry. It should be warned, however, that an embedding of a graph grammar into a physical environment may not respect the rules of the grammar. For example, the geometries of the parts and the environment may render reachable assemblies in the grammar impossible to realize physically. The process of embedding a grammar in a physical system is more intuitive than formal, and is the main focus of our present research.

Acknowledgments

The author's research in self-assembly is supported in part by NSF Grant #0347955. The author wishes to thank his collaborators Rob Ghrist and Steve Lipsky at UIUC.

References

1. N. L. Abbott, C. B. Gorman, and G. M. Whitesides. Active control of wetting using applied electrical potentials and self-assembled monolayers. *Langmuir*, 11(1):16–18, 1995.
2. L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computin*, pages 740–748, Hersonissos, Greece, 2001.
3. L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. Wilhelm P. Moisset de Espanés, and K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 23–32, Montreal, Canada, May 2002.
4. B. Berger, P.W. Shor, L. Tucker-Kellogg, and J. King. Local rule-based theory of virus shell assembly. *Proceedings of the National Academy of Science, USA*, 91(6):7732–7736, August 1994.
5. K. F. Böhringer, M. Cohn, K. Y. Goldberg, R. Howe, and A. Pisano. Parallel microassembly with electrostatic force fields. In *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 1998.
6. B. Bollobás. *Modern Graph Theory*. Springer, 1991.

7. B. Courcelle. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, chapter on Graph Rewriting: An Algebraic and Logic Approach, pages 193–242. MIT Press, 1990.
8. H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69, 1979.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
10. C. Jones and M. J. Matarić. From local to global behavior in intelligent self-assembly. In *International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.
11. E. Klavins. Automatic synthesis of controllers for distributed assembly and formation forming. In *Proceedings of the IEEE Conference on Robotics and Automation*, Washington DC, May 2002.
12. E. Klavins. Automatically synthesized controllers for distributed assembly: Partial correctness. In S. Butenko, R. Murphey, and P. M. Pardalos, editors, *Cooperative Control: Models, Applications and Algorithms*, pages 111–127. Kluwer, 2002.
13. E. Klavins. Toward the control of self-assembling systems. In A. Bicchi, H. Christensen, and D. Prattichizzo, editors, *Control Problems in Robotics*, pages 153–168. Springer Verlag, 2002.
14. E. Klavins, R. Ghrist, and D. Lipsky. Graph grammars for self-assembling robotic systems. In *Proceedings of the International Conference on Robotics and Automation*, 2004. To Appear.
15. W. K. Rothemund. Using lateral capillary forces to compute by self-assembly. *Proceedings of the National Academy of Science, USA*, 97(3):984–989, 2000.
16. K. Saitou. Conformational switching in self-assembling mechanical systems. *IEEE Transactions on Robotics and Automation*, 15(3):510–520, 1999.
17. K. Saitou and M. Jakiela. Automated optimal design of mechanical conformational switches. *Artificial Life*, 2(2):129–156, 1995.
18. Y. S. Smentanich, Y. B. Kazanovich, and V. V. Kornilov. A combinitorial approach to the problem of self assembly. *Discrete Applied Mathematics*, 57:45–65, 1995.
19. R. L. Thompson and N. S. Goel. Movable finite automata (MFA) models for biological systems I: Bacteriophage assembly and operation. *Journal of Theoretical Biology*, 131:152–385, 1988.
20. H. Wang. Notes on a class of tiling problems. *Fundamenta Mathematicae*, pages 295–305, 1975.
21. 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.