Proceedings of the 2008 IEEE
International Conference on Robotics and Biomimetics
Bangkok, Thailand, February 21 - 26, 2009

Coordinated Three-Dimensional Robotic Self-Assembly

Jonathan Kelly
Department of Computer Science
University of Southern California
Los Angeles, California, USA 90089-0781
jonathsk@usc.edu

Hong Zhang
Department of Computer Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E1
zhang@cs.ualberta.ca

***Abstract*—Nature has demonstrated that geometrically interesting and functionally useful structures can be built in an entirely distributed fashion. We present a biologically-inspired model and several algorithms for three-dimensional self-assembly, suitable for implementation by very simple reactive robots. The robots, which we call assembly components, have limited local sensing capabilities and operate without centralized control. We consider the problem of maintaining coordination of the assembly process over time, and introduce the concept of an assembly ordering to describe constraints on the sequence in which components may attach to a growing structure. We prove the sufficient properties of such an ordering to guarantee production of a desired end result. The set of ordering constraints can be expressed as a directed acyclic graph; we develop a graph algorithm that is able to generate ordering constraints for a wide variety of structures. We then give a procedure for encoding the graph in a set of local assembly rules. Finally, we show that our previous results for the optimization of rule sets for two-dimensional structures can be readily extended to three dimensions.**

***Index Terms*—Self-assembly, multi-robot coordination, graph theory, optimization, randomized algorithms.**

I. Introduction

In this paper, we describe a model and several associated algorithms for three-dimensional self-assembly. We specifically consider the problem of enabling the *coordinated* assembly of complex structures using very simple agents. This extends our previous work [1] to three dimensions and to a larger class of structures. Self-assembly, in this context, is defined as a process in which the agents involved *become part of the final structure* being built.

If we wish to employ self-assembly for our own purposes, we must immediately deal with two fundamental issues: maintaining coordination of the assembly process, and ensuring that a desired structure is produced reliably. Some form of coordination is required to prevent the agents from producing a disorganized or random result. The reliability of the assembly process, in turn, depends on the agents' ability to correctly identify some set of environmental features – for agents with limited sensing capabilities, identification becomes more difficult as the size of this set increases.

In many large-scale systems, the coordination problem is particularly challenging because agents are able to interact with each other only *locally*. Long-range sensing and communication are either prohibitively costly or impossible. A challenge, then, is to solve the *local-to-global problem*: given a system of many agents interacting locally, how does one program the system to produce a specific *global* result?

To explore these issues from an algorithmic perspective, we introduce a model in which unit-cube *assembly components* (our agents) move on a discrete lattice and use simple rules to self-assemble into three-dimensional structures. First, we formally define a set of sufficient local spatiotemporal ordering constraints that enable assembly to progress in a coordinated manner. Production of a desired structure is *guaranteed* as long as these constraints are respected. We develop an algorithm for expressing the constraints as a directed acyclic graph, and show that the algorithm can correctly describe constraints for complex structures. We then give a procedure for encoding the graph in a set of local assembly rules. By doing so, we solve an instance of the local-to-global problem – the rules produce implicit coordination among agents and require no direct agent-to-agent communication. Finally, we extend our randomized optimization algorithm, originally presented in [1], to three dimensions. The algorithm attempts to minimize the number of unique environmental features that appear in the assembly rules, by iteratively refining a worst-case solution.

The remainder of the paper is organized as follows. We review prior work in Section II. Our assembly model is introduced in Section III, and we examine related coordination issues in Section IV. We then briefly describe our randomized optimization algorithm in Section V. Experimental results are presented in Section VI. We offer some conclusions and directions for future research in Section VII.

II. Prior and Proximal Work

Relevant prior and proximal work falls broadly into two categories: *self-assembly* and *collective assembly*. Although our focus here is on self-assembly, we discuss several important results for collective systems below as well.

Self-assembly can be viewed as a form of computation, where the input to a self-assembly 'program' is a set of individual atomic (irreducible) components or parts, and the output is a final, aggregate structure. In [2], for example,

Adleman presents a model for the one-dimensional self-assembly of linear chains of square planar tiles. The edges of each tile are assigned integer *glue* values; tiles will bind along abutting edges if their glues are compatible. Adleman provides a metric for the time complexity of chain formation based on counting the number of bindings between tiles.

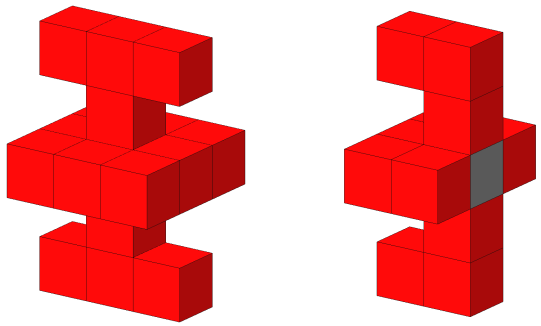
Rothemund and Winfree [3] describes a two-dimensional *Tile Assembly Model*, in which unit-square tiles move randomly on the plane and join together to form larger structures. A *tile type* is a unique assignment of glue values (cf. [2]) to the edges of a tile. A tile will bind with one or more neighbors if the total strength (sum) of their pairwise edge interactions, defined by a *glue strength function*, is larger than a fixed temperature parameter. Work in [4] gives an assembly algorithm for $n \times n$ squares that is optimal in terms of the required number of tile types and the asymptotic assembly time. The problem of determining whether a certain number of tile types is the minimum needed to uniquely assemble an arbitrary planar structure is shown to be NP-complete in [5].

Klavins et al. [6] presents a graph-based model for robotic self-assembly, where graph vertices represent parts and edges between vertices indicate that the corresponding parts are attached. Assembly rules are specified by a grammar consisting of a set of graph pairs: if a conformation of parts matching the first graph in a pair exists, the conformation can be replaced with a new conformation defined by the second graph. Ghrist and Lipsky [7] extends graph grammars to tile assembly systems, and gives examples of grammars that generate only planar outputs, but does not address the problem of designing grammars for arbitrary planar structures.

Closely related to our own work is the *Intelligent Self Assembly* (ISA) model proposed by Jones and Matarić [8], in which autonomous unit-square Assembly Agents (AAs) self-assemble into planar structures under local, rule-based control. Rules are generated offline by a Transition Rule Set (TRS) compiler, which takes the goal structure as an input and produces a set of assembly rules as an output. The compiler is able to generate assembly rules for a limited subset of planar structures only, however.

Arbuckle and Requicha [9] proposes a *communicative* model for self-assembly, involving agents that are able to send messages to connected neighbours. Communication enables a global ordering to be imposed on the assembly process, and facilitates adaptation to unknown obstacles in the environment. The disadvantage of communication is that it requires significant resources (e.g. power) from each assembly agent. For this reason, we specifically develop a system in which explicit communication is not required.

A relevant example of collective assembly is the model of wasp nest construction proposed by Bonabeau et al. [10]. Here, reactive agents move randomly and asynchronously on a three-dimensional lattice, depositing elementary bricks when they sense certain *stimulating configurations* of bricks



(a) (b)

Fig. 1: (a) Structure A, composed of 17 assembly components. (b) Cutaway view, showing the seed component in gray. The seed is always positioned in cell $(0, 0, 0)$.

in adjacent cells. There is no centralized control or direct communication in the system, and agents have access only to local information about the portion of a structure in their immediate vicinity. The rule space in the model is extremely large – however, there are relatively few subsets of rules which generate organized architectures [11]. Work in [12] explores the rule space using a genetic algorithm (GA), where the fitness function is based on subjective human evaluation of the 'structuredness' of a series of example architectures. Although the structures produced by the GA are visibly organized, [12] does not consider the problem of generating rules that assemble specific structures in an exact and repeatable way. In contrast, we are able to produce deterministic rules that lead to a pre-specified end result.

Werfel et al. suggest an approach called *extended stigmergy*, using a bipartite system of blocks and robots, for two-dimensional [13] and three-dimensional [14] collective assembly. Square or cubic structural blocks communicate state information to their neighbours and to nearby mobile robots. Using this information, the robots are able to assemble a range of two- and three dimensional structures, specified by either a complete design or by a set of more general spatial constraints. This bipartite approach is more complicated than our homogeneous model, and has the same drawbacks as those outlined above for [9].

III. A Model for Three-Dimensional Self-Assembly

Our assembly model uses a simple discretization of time and space, in which a number of assembly components independently traverse a three-dimensional Cartesian lattice. A position in the lattice, specified by a unique coordinate triple (x, y, z), is called a *cell*. Each cell has six adjacent neighbors to the north, east, south and west, and below and above. Our goal is to produce a specific *target* $\mathcal{T}$, or a set of cells that, when occupied, form a desired shape. The set $\mathcal{T}$ always includes the origin cell, $(0, 0, 0)$. A structure is

complete when it contains exactly the same set of occupied cells as the target.

The assembly process begins with a group of $n \geq 1$ *free components* at a set $\mathcal{F}$ of random lattice positions, and a single *seed component*, which is fixed at the origin cell. Growth of a structure occurs outwards from the seed. We assume that the lattice is of sufficient size to accommodate the target that we wish to build, with an additional border of cells (at least one unit wide) around the entire perimeter.

At each discrete time step, every free component performs a random walk, moving to an adjacent empty cell if possible. As a component moves, it compares the occupancy pattern in the six adjacent cells with entries in an internal lookup table of *assembly rules*. An assembly rule is defined by a local *binding configuration* of assembly components in the adjacent cells, and an identifier (feature or state) associated with each component; we use the term *label* for such an identifier. For our purposes, labels will be values from the set $\mathbb{N}^+$ of positive integers. In a physical system, a label might be any characteristic which is readily discernible at the appropriate scale. At least one cell in a binding configuration must be occupied, and up to five cells may be occupied.

When a binding configuration is identified, the component *applies* the corresponding rule by attaching (binding) itself to the adjacent components at its current lattice position. Binding is irreversible – assembly actions are never undone. The component is then *bound* to the structure, and the number of free components is reduced by one. Immediately after binding, the component performs a *label update*, transitioning from the null label ('0' by our convention) to the *resultant label* defined by the activated rule. The rules are identical for all components, making each component redundant and interchangeable.

An assembly component is a simple, reactive robot with minimal sensing capabilities: it is able to determine its orientation within the lattice, and to identify the labels on other components in adjacent, occupied cells. The components have no knowledge of their own lattice positions, and no capacity to acquire or compute this information. Additionally, components have no memory of past actions or observations and do not communicate directly with one another.

Formally, we define a structure as a set $\mathcal{S}$ of occupied lattice cells, identified by their coordinates. A *label set* is a set $\mathcal{L} \subset \mathbb{N}^+$, with $|\mathcal{L}|$ finite. We model the assembly of *connected* structures only, in which there is a path of adjacent cells between every two cells in $\mathcal{S}$.

A binding configuration is an ordered six-tuple $(l_n, l_e, l_s, l_w, l_d, l_u)$ of values from the set $\mathbb{N}^+ \cup \{-\}$, where the values represent labels on components in the six adjacent cells to the north, east, south, west and below and above. The symbol '$-$' is used to designate cells in the configuration that may either be empty or be occupied by a free component.

The assembly process yields a labeled structure $(\mathcal{S}, \mathcal{L}, \lambda)$, where $\mathcal{S}$ is a (connected) structure, $\mathcal{L}$ is a label set, and λ is a surjection $\lambda : \mathcal{S} \to \mathcal{L}$. We will identify a labeled structure by a set of *(coordinates, label)* pairs, or, for brevity, by a symbol $\bar{\mathcal{S}}$ with an over-bar. The seed component is always given the initial label '1'.

An assembly rule is an ordered seven-tuple $r = (l_n, l_e, l_s, l_w, l_d, l_u, \gamma)$, where the first six values define the binding configuration and $\gamma \in \mathcal{L}$ is the resultant label. Rule r may be applied by a free component in cell (i, j, k) if and only if the binding configuration for r exactly matches the current lattice configuration at (i, j, k). When r is applied, we update the set $\mathcal{F}$ of positions of free components as $\mathcal{F} \leftarrow \mathcal{F} \setminus \{(i, j, k)\}$ and the labeled structure $\bar{\mathcal{S}}$ as $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \{((i, j, k), \gamma)\}$.

IV. Coordination of the Assembly Process

Assembly cannot occur in an entirely arbitrary way – components must already be positioned in certain lattice cells before other components may bind to the growing structure. Further, since binding is irreversible, it is necessary to ensure that groups of components do not prematurely form barriers which prevent free components from moving to fill vacant target cells.

Below, we introduce the concept of an *assembly ordering* to describe these constraints. We begin by noting that each lattice cell is always in one of three mutually exclusive states: accessible, inaccessible, or occupied by a bound component. The empty cell (i, j, k) is *accessible* if and only if, with all free components removed from the lattice and starting at an empty cell on the perimeter, there is a connected set of pairwise-adjacent empty cells that includes (i, j, k). Otherwise, the empty cell is *inaccessible*, and free components will be unable to reach the cell.

A. Ordering Constraints

An *assembly ordering* for a target $\mathcal{T}$ is a set of temporal ordering constraints over the cells in $\mathcal{T}$. Let $\prec$ be the binary *predecessor* relation, such that for any two cells $(i, j, k), (p, q, r) \in \mathcal{T}$, $(i, j, k) \prec (p, q, r)$ if cell (i, j, k) must be occupied by a bound component before cell (p, q, r) may be occupied. We call (i, j, k) a *predecessor cell* (or simply *predecessor*) of (p, q, r), and (p, q, r) a *successor cell* (or simply *successor*) of (i, j, k). The seed cell is a predecessor of every other cell in $\mathcal{T}$. Together, the set $\mathcal{T}$ and relation $\prec$ define an assembly ordering $(\mathcal{T}, \prec)$.

The ordering $(\mathcal{T}, \prec)$ for target $\mathcal{T}$ is *valid* if and only if, for any assembly sequence that respects $(\mathcal{T}, \prec)$ and at any stage of the assembly process, every cell in $\mathcal{T}$ is either accessible or occupied by a bound component – that is, if assembly is always able to proceed to completion. We say that an ordering is *violated* when an assembly rule allows a component to bind to a structure before one of the predecessor cells is occupied.

Algorithm 1 Breadth-First Assembly Graph for Target

Input: Target $\mathcal{T}$
Output: Breadth-first directed assembly graph G'

$V \leftarrow$ set of vertices, one vertex for each cell in $\mathcal{T}$
$E \leftarrow$ set of undirected edges, one edge for every pair of adjacent cells in $\mathcal{T}$
$G \leftarrow (V, E)$ *// undirected adjacency graph*
$E' \leftarrow \emptyset$
$open \leftarrow \{v_{(0,0,0)}\}$ *// FIFO queue*
$closed \leftarrow \emptyset$
while $open$ is not empty **do**
  $v_{(i,j,k)} \leftarrow$ dequeued vertex from front of $open$
  $closed \leftarrow closed \cup \{v_{(i,j,k)}\}$
  for each vertex $v_{(p,q,r)}$ adjacent to $v_{(i,j,k)}$ in G **do**
    if $v_{(p,q,r)} \notin closed$ **then**
      if $v_{(p,q,r)} \notin open$ **then**
        add $v_{(p,q,r)}$ to end of $open$
      end if
    else
      $E' \leftarrow E' \cup \{(v_{(p,q,r)}, v_{(i,j,k)})\}$ *// add edge*
    end if
  end for
end while
return $G' = (V, E')$

To define a valid ordering, we will divide the set of cells in the target into two disjoint subsets: *exterior boundary cells* and *interior cells*. Consider a lattice containing a complete target $\mathcal{T}$, with all other cells empty. Choose a cell (s, t, u) outside of the bounding box that encloses $\mathcal{T}$. A cell $(i, j, k) \in \mathcal{T}$ is an *exterior boundary cell* of $\mathcal{T}$ if, with (i, j, k) empty and all other cells in $\mathcal{T}$ occupied, (i, j, k) is accessible from (s, t, u). Likewise, let $\mathcal{T}$ be a target, and $\mathcal{B} \subseteq \mathcal{T}$ be the exterior boundary set for $\mathcal{T}$. An *interior cell* is a cell $(p, q, r) \in \mathcal{T} \setminus \mathcal{B}$.

With these definitions in hand, we now state an important theorem about valid assembly orderings. An abbreviated proof of the theorem is given in the appendix.

Theorem 1 An assembly ordering $(\mathcal{T}, \prec)$ is valid if and only if, for each cell in $\mathcal{T}$ except the seed, there is at least one adjacent predecessor cell, and for each interior cell there is at least one adjacent successor cell.

B. Assembly Graphs

An assembly ordering can be expressed as a directed acyclic graph, consisting of a set of vertices, corresponding to the cells in the target, and a set of directed edges, corresponding to pairwise spatiotemporal ordering constraints between the cells. We call such a graph an *assembly graph*; this formalism will allow us to use a breadth-first graph traversal algorithm to produce valid assembly orderings for a large class of structures. The assembly graph algorithm requires an initial, undirected adjacency graph for a target $\mathcal{T}$. We form this adjacency graph $G = (V, E)$ by:

- adding a vertex $v_{(i,j,k)}$ to the set V for each cell in $(i, j, k) \in \mathcal{T}$, and
- adding an undirected edge $(v_{(i,j,k)}, v_{(p,q,r)})$ to the set E for every pair of adjacent cells $(i, j, k), (p, q, r) \in \mathcal{T}$.

Algorithm 2 Rule Set from Labeled Structure

Input: Labeled structure $\bar{\mathcal{S}}$ and assembly ordering $(\mathcal{T}, \prec)$
Output: Rule set $\mathcal{R}$

$\mathcal{R} \leftarrow \emptyset$
for each cell $(i, j, k) \in \mathcal{T}$ **do**
  for cell (p, q, r) adjacent to (i, j, k) in direction $d \in \{n, e, s, w, d, u\}$ **do**
    if $(p, q, r) \in \mathcal{T}$ and $(p, q, r) \prec (i, j, k)$ **then**
      $l_d \leftarrow \bar{\mathcal{S}}(p, q, r)$
    else
      $l_d \leftarrow$ '-'
    end if
  end for
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{(l_n, l_e, l_s, l_w, l_d, l_u, \bar{\mathcal{S}}(i, j, k))\}$
end for
return $\mathcal{R}$

Using the undirected graph G, which defines the static relationships between components in the final structure, we can construct a new, directed graph G' which contains information about the dynamic relationships that exist as the structure evolves from the seed.[1] We form a directed assembly graph G' for $\mathcal{T}$ from the same set of vertices V, by adding a directed edge from vertex $v_{(i,j,k)}$ to vertex $v_{(p,q,r)}$ if the corresponding cell (i, j, k) is adjacent to cell (p, q, r) and (i, j, k) is a predecessor of (p, q, k). A sink vertex in the graph represents a cell with no successors. There is a single source vertex in the graph corresponding to the seed cell. Note also that we can associate a set of *exterior boundary vertices* (resp. *interior vertices*) in V with the set of exterior boundary cells (resp. interior cells) in $\mathcal{T}$.

We now consider the problem of generating valid assembly orderings for specific targets. In general, there may be many valid orderings for a given target; the algorithm presented here generates a *breadth-first* assembly graph and ordering. The assembly graph is built by traversing the target's adjacency graph breadth-first, starting from the seed.

Algorithm 1 is a modified version of the standard breadth-first graph traversal. As we encounter each vertex in the undirected graph G, we incrementally build a corresponding assembly graph G' by adding a set of directed edges. When we arrive at a vertex v, we add directed edges from all

[1] We will use the prime marker, $'$, to indicate that a graph is directed.

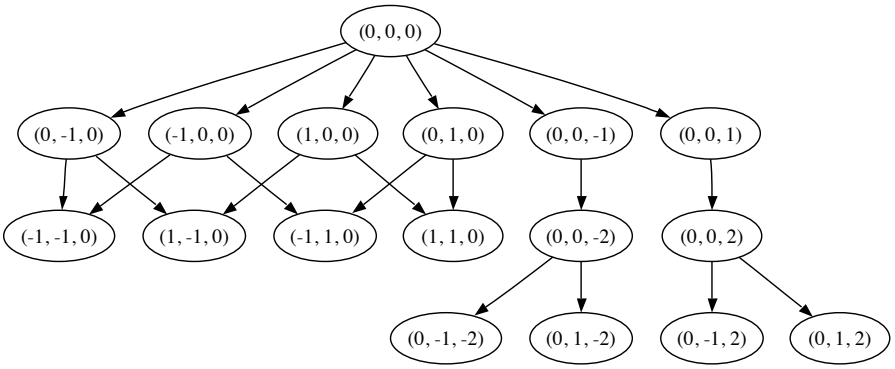


Fig. 2: Breadth-first assembly graph for structure shown in Figure 1. The graph was generated by Algorithm 1. Each node corresponds to a lattice cell; edges in the graph indicate predecessor-successor relationships.

adjacent, previously-visited vertices to the set E'.

To determine if the graph produced by Algorithm 1 represents a valid ordering, we can directly apply Theorem 1. The algorithm adds a directed edge between every adjacent pair of vertices in G, and the breadth-first traversal ensures that G' is acyclic. If all sink vertices in G' are exterior boundary vertices, then the graph represents a valid ordering. Conversely, if an interior sink vertex exists, then the graph cannot represent a valid ordering because there is at least one interior vertex without an adjacent successor. As an example, the (valid) assembly graph generated by Algorithm 1 for Structure A is shown in Figure 2.

C. From Graph to Assembly Rules

Once we have a valid assembly graph $G' = (V, E')$ for $\mathcal{T}$, we can compile a set of local assembly rules from the graph. We first produce the set of pairwise ordering constraints (which define the assembly ordering $(\mathcal{T}, \prec)$) by visiting every vertex $v \in V$, adding a single constraint for each edge that is incident on v.

To build the rule set, we begin with an empty lattice, and place a component with a unique label at each position in $\mathcal{T}$, creating a labeled structure $\bar{\mathcal{S}}$. Then, as described by Algorithm 2, we step through the assembly ordering cell by cell, adding rules sequentially to the rule set. For a cell (i, j, k), we add a rule whose binding configuration is defined by the labels on components in adjacent predecessor cells (with non-predecessor entries set to '-'), and using the resultant label already assigned to the component in cell (i, j, k).

The drawback of this procedure, described by Algorithm 3, is that $|\mathcal{T}|$ labels and $|\mathcal{T}| - 1$ rules are always required for a $|\mathcal{T}|$-cell target. We call such a $|\mathcal{T}|$-label rule set the *worst-case* rule set for $\mathcal{T}$, and use this worst-case solution as the input to the optimization algorithm described in the next section.

Algorithm 3 Worst-Case Rule Set

Input: Target $\mathcal{T}$ and valid ordering $(\mathcal{T}, \prec)$
Output: Consistent rule set $\mathcal{R}$

1: $\bar{\mathcal{S}} \leftarrow$ structure produced by placing a uniquely-labeled component at each cell in $\mathcal{T}$
2: $\mathcal{R} \leftarrow$ rule set generated by Alg. 2 from $(\bar{\mathcal{S}}, (\mathcal{T}, \prec))$
3: **return** $(\bar{\mathcal{S}}, \mathcal{R})$

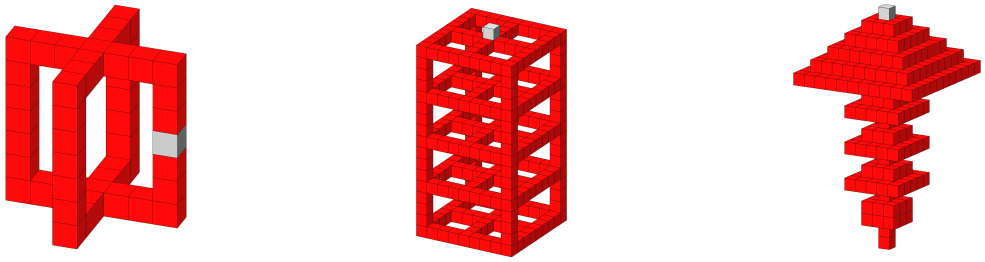
V. Label Set Optimization

For physically very simple assembly components, the sensing fidelity required to correctly recognize a large number of distinct labels may be difficult to achieve in practice. However, it is often possible to exploit symmetry and self-similarity within a structure to compress the size of the label set. This naturally leads to two related combinatorial optimization problems:

- The **(Full) Minimum Label Set Problem:** Given a target $\mathcal{T}$, find a valid assembly ordering $(\mathcal{T}, \prec)$ and rule set $\mathcal{R}$ such that $\mathcal{R}$ is consistent for $\mathcal{T}$ under $(\mathcal{T}, \prec)$ and uses the minimum number of unique labels possible.
- The **Restricted Minimum Label Set Problem:** Given a target $\mathcal{T}$ and a valid assembly ordering $(\mathcal{T}, \prec)$, find a rule set $\mathcal{R}$ such that $\mathcal{R}$ is consistent for $\mathcal{T}$ under $(\mathcal{T}, \prec)$ and uses the minimum number of unique labels possible.

In [1], we proposed an algorithm called *randomized* (or *stochastic*) *contraction* as an approximate solution for the Restricted MLS problem. We briefly describe the algorithm below; a more complete exposition is given in [1] and [15].

The contraction algorithm operates by attempting to iteratively reduce (contract) the size of a label set $\mathcal{L}$. Given a target $\mathcal{T}$ and a valid assembly ordering, we generate a worst-case rule set $\mathcal{R}$ and the associated labeled structure $\bar{\mathcal{S}}$ using Algorithm 3. Then, at each iteration, we randomly select a component in $\bar{\mathcal{S}}$, change the label on the component to a random value between 1 and the existing value, and generate



(a) Structure B (b) Structure C (c) Structure D

Fig. 3: a) Structure B, composed of 46 assembly components, b) Structure C, composed of 274 assembly components, and c) Structure D, composed of 412 assembly components. The seed components are shown in gray.

an updated rule set $\mathcal{R}^*$ using Algorithm 2. If $\mathcal{R}^*$ assembles $\mathcal{T}$ only, the change is accepted, otherwise the change is rejected. This process continues for a certain number of iterations, or until there is no further improvement (reduction in the label count).

It is possible to verify in polynomial time whether or not a candidate rule set $\mathcal{R}^*$ assembles $\mathcal{T}$ only (while respecting the assembly ordering). We presented an algorithm in [1] which performs this *consistency* check; we give only a short description here. The algorithm operates by maintaining a set $\mathcal{K}$ of empty cells that form part of the *frontier* of the growing structure $\bar{\mathcal{S}}$. The frontier set is the set of cells adjacent to occupied cells already in $\bar{\mathcal{S}}$, at which a binding event could occur at the next time step. At each assembly step and for every position in $\mathcal{K}$, the algorithm determines if more than one rule can potentially be applied, or if there is a rule which would add a component at a position not in $\mathcal{T}$. If either condition is true, the rule set does not uniquely assemble $\mathcal{T}$. Otherwise, the algorithm iteratively adds components to the labeled structure $\bar{\mathcal{S}}$ until no further rules can be applied. If, at that time, $\bar{\mathcal{S}}$ has the same number of occupied cells as $\mathcal{T}$, then $\mathcal{R}$ is consistent for $\mathcal{T}$. The consistency check requires $O(|\mathcal{R}||\mathcal{T}|)$ time.

VI. Experiments

We performed a series of experiments to verify the operation of the ordering and optimizations algorithm for the example structures shown in Figures 1 and 3. Optimization results are given in Table I. The table also lists the number of assembly steps for each structure. This is the number of steps required to assemble the entire target, if all possible binding sites on the exterior of the growing structure are filled synchronously at every step. The metric corresponds to the length of the longest path in the assembly graph, and gives some indication of the overall assembly time.

The results in Table I show that, in all cases, we are able to significantly reduce the number of labels required to assemble each structure (compared to the worst case). For Structure A, it is possible to verify by brute force that the rule set is optimal by our criterion, using the minimal number of labels possible. The optimized rule set for Structure A is listed in Table II. In the case of Structure D, which appeared previously in [12], we are able to produce the exact structure *deterministically*, instead of with some statistical probability, at the expense of using more labels than in [12].

VII. Conclusions and Future Work

In this paper, we extended our previous work on planar distributed assembly to three dimensions. Our primary contribution was a group of algorithms for generating *local* assembly rules which are guaranteed to produce a desired *global* result (structure), despite the limited capabilities and random actions of the agents involved. This result, coupled with our label set optimization algorithm, enables the production of compact sets of assembly rules for a wide variety of structures.

We are pursuing a number of directions for future work. Currently, we are performing a series of simulation studies to characterize the relationship between the choice of assembly graph and the average assembly time. We are also exploring ways to further reduce the number of labels required to assemble a structure, by purposely using an under-constrained rule set and evaluating the outcome in terms of the statistical percentage yield of the desired end product. Ultimately, we hope to build a physical implementation of the model we have described.

Acknowledgements

This research was funded in part by grants from the Government of Canada (NSERC) and the Government of Alberta (iCORE).

Appendix: Proof of Theorem 1

Proof: Our connectivity constraint requires that *every* cell except the seed have at least one adjacent predecessor, so the first clause of Theorem 1 is immediately satisfied. Further, cells in the exterior boundary set for $\mathcal{T}$ are always accessible (assuming that components only bind to the structure at positions in $\mathcal{T}$)[2], and can therefore be removed from

[2]We show how to generate assembly rules which enforce this in [15].

TABLE I: Optimization Results for Structures A through D

Structure	Rules	Labels	Steps	Iterations
A (17 components)	14	**4**	3	55
B (46 components)	35	**23**	12	1,985
C (274 components)	189	**114**	33	24,607
D (412 components)	289	**41**	19	17,953

consideration. So we must now show that an ordering is valid if and only if each interior cell has an adjacent successor cell. We prove by contradiction.

Consider an interior cell (i, j, k), and assume that the assembly ordering is valid but that (i, j, k) has no successors. If (i, j, k) has no successors, then we must be able to assemble the remaining components in $\mathcal{T}$ regardless of whether cell (i, j, k) is occupied or not – there are no constraints in $(\mathcal{T}, \prec)$ that prevent this. Assume that (i, j, k) is empty and, according to $(\mathcal{T}, \prec)$, we place components in the remaining cells in $\mathcal{T}$. Cell (i, j, k) is an interior cell, and the final structure is connected, so there must exist a subset of occupied cells $Q \subset \mathcal{T}$ that block access to (i, j, k) – otherwise (i, j, k) would be an exterior boundary cell. But if $(\mathcal{T}, \prec)$ is valid, then such a subset cannot exist, and we have reached a contradiction.

Conversely, assume that every interior cell in $\mathcal{T}$ has an adjacent successor cell, but that the assembly ordering is not valid. Consider an inaccessible interior cell (i, j, k), and let $Q \subset \mathcal{T}$ be a set of occupied cells that block access to (i, j, k). Let W be the set containing (i, j, k) and any successors of (i, j, k) that are also blocked by Q. There must exist a cell $(p, q, r) \in W$ that does not have an adjacent successor, since W is finite and the predecessor relation is anti-symmetric.[3] According to $(\mathcal{T}, \prec)$, (p, q, r) must therefore be an exterior boundary cell. But if (p, q, r) is an exterior boundary cell, it is always accessible, and cannot be blocked by Q. We have again reached a contradiction, which completes the proof. ■

REFERENCES

[1] J. Kelly and H. Zhang, "Combinatorial Optimization of Sensing for Rule-Based Planar Distributed Assembly," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS'06)*, Beijing, China, Oct. 2006, pp. 3728–3734.

[2] L. Adleman, "Towards a Mathematical Theory of Self-Assembly," Department of Computer Science, University of Southern California, Los Angeles, USA, Tech. Rep. 00-722, Jan. 2000.

[3] P. Rothemund and E. Winfree, "The Program-size Complexity of Self-assembled Squares," in *Proc. Thirty-Second Ann. ACM Symp. Theory of Computing (STOC'00)*, Portland, USA, May 2000, pp. 459–468.

[4] L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang, "Running Time and Program Size for Self-assembled Squares," in *Proc. Thirty-Third Ann. ACM Symp. Theory of Computing (STOC'01)*, Hersonissos, Greece, July 2001, pp. 740–748.

[3] We prove this property of the predecessor relation in [15].

TABLE II: Optimized rule set for Structure A

Rule #	Binding Configuration (n, e, s, w, d, u)	Resultant Label
1	(0, 0, 0, 0, 0, 1)	3
2	(0, 0, 0, 0, 0, 3)	4
3	(0, 0, 0, 0, 1, 0)	3
4	(0, 0, 0, 0, 3, 0)	4
5	(0, 0, 0, 1, 0, 0)	2
6	(0, 0, 0, 4, 0, 0)	2
7	(0, 0, 1, 0, 0, 0)	2
8	(0, 0, 2, 2, 0, 0)	2
9	(0, 1, 0, 0, 0, 0)	2
10	(0, 2, 2, 0, 0, 0)	2
11	(0, 4, 0, 0, 0, 0)	2
12	(1, 0, 0, 0, 0, 0)	2
13	(2, 0, 0, 2, 0, 0)	2
14	(2, 2, 0, 0, 0, 0)	2

[5] L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund, "Combinatorial Optimization Problems in Self-Assembly," in *Proc. Thirty-Fourth Ann. ACM Symp. Theory of Computing (STOC'02)*, Montréal, Canada, May 2002, pp. 23–32.

[6] E. Klavins, "Directed Self-Assembly Using Graph Grammars," in *Foundations of Nanoscience: Self Assembled Architectures and Devices*, Snowbird, USA, Apr. 2004.

[7] R. Ghrist and D. Lipsky, "Grammatical Self Assembly for Planar Tiles," in *Proc. IEEE Int'l Conf. MEMS, NANO, and Smart Systems (ICMENS'04)*, Banff, Canada, Aug. 2004, pp. 205–211.

[8] C. V. Jones and M. J. Matarić, "From Local to Global Behavior in Intelligent Self-Assembly," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA'03)*, Taipei, Taiwan, Sept. 2003, pp. 721–726.

[9] D. Arbuckle and A. A. G. Requicha, "Active Self-Assembly," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA'04)*, vol. 1, New Orleans, USA, Apr. 2004, pp. 896–901.

[10] E. Bonabeau, G. Theraulaz, E. Arpin, and E. Sardet, "The Building Behavior of Lattice Swarms," in *Proc. Fourth Int'l Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds., Artificial Life IV. Cambridge, USA: MIT Press, 1994, pp. 307–312.

[11] G. Theraulaz and E. Bonabeau, "Coordination in Distributed Building," *Science*, vol. 269, no. 5224, pp. 686–688, Aug. 1994.

[12] E. Bonabeau, S. Guéin, D. Snyers, P. Kuntz, and G. Theraulaz, "Three-dimensional architecture grown by simple 'stigmergic' agents," *Biosystems*, vol. 56, no. 1, pp. 13–32, 2000.

[13] J. Werfel, Y. Bar-Yam, and R. Nagpal, "Building Patterned Structures with Robot Swarms," in *Proc. Nineteenth Int'l Joint Conf. Artificial Intelligence (IJCAI '05)*, Edinburgh, Scotland, Aug. 2005, pp. 1495–1502.

[14] J. Werfel and R. Nagpal, "Three-Dimensional Construction with Mobile Robots and Modular Blocks," *Int'l J. Robotics Research*, vol. 27, no. 3–4, pp. 463–479, Mar./Apr. 2008.

[15] J. Kelly, "Algorithmic Distributed Assembly," Master's thesis, University of Alberta, Edmonton, Canada, Apr. 2008.