Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Combinatorial Optimization of Sensing for Rule-Based Planar Distributed Assembly

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

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

***Abstract*— We describe a model for planar distributed assembly, in which agents move randomly and independently on a two-dimensional grid, joining square blocks together to form a desired target structure. The agents have limited capabilities, including local sensing and rule-based reactive control only, and operate without centralized coordination. We define the spatiotemporal constraints necessary for the ordered assembly of a structure and give a procedure for encoding these constraints in a rule set, such that production of the desired structure is guaranteed. Our main contribution is a stochastic optimization algorithm which is able to significantly reduce the number of environmental features that an agent must recognize to build a structure. Experiments show that our optimization algorithm outperforms existing techniques.**

I. Introduction

In this paper we investigate distributed assembly, the process by which a group of agents interact to assemble a coherent structure or pattern from individual components. We define an agent as an autonomous entity capable of recognizing and responding to some set of features within its environment. Nature has demonstrated the utility of distributed assembly at a range of spatial scales, from the molecular machinery inside living cells to the to highly organized nest-building activities of many social insect species.

The distribution of an assembly task is potentially advantageous for several reasons, among them the improvements in performance and failure tolerance that can result from parallelism and redundancy (respectively). However, systems containing hundreds or thousands of agents are frequently *locally-interacting* only – the cost of communication, in terms of power, time, or complexity, between distant agents is often too high to be practical.

In an effort to understand how such locally-interacting systems can be programmed to produce useful artifacts, several groups have introduced rule-based abstractions of physical assembly processes. Typically, the agents in these abstractions possess restricted, short-range sensing and operate without centralized control. Assembly involves joining tiles, blocks or discs together to form larger aggregate structures, with coordination enforced by the rules that govern the actions of the agents. The rules are if–then statements, either explicitly given or implicit in the model, which describe the local environmental states that prompt an assembly action.

Previous research has focused on the problem of designing assembly rules such that the agent-environment interactions lead to the desired global result. We examine a different but related issue. The efficiency and accuracy of the assembly process depends on an agent's ability to correctly recognize and discriminate between members of a set of environmental features – for agents with limited sensing capabilities, discrimination becomes more difficult as the number of features increases. How can we minimize the number of distinct features that an agent must recognize in order to assemble a given structure?

To answer this question, we propose a distributed assembly model in which minimalist agents use local rules to assemble structures composed of square blocks. Our goal is to reduce the required number of features, which we call *labels*, that an agent must recognize, while still ensuring that the desired structure is produced. This goal is made more difficult because our agents move randomly, do not communicate directly with each other, and maintain no history of past actions or observations. We show how to generate sets of rules that, when executed by the agents, deterministically produce the desired result, despite the random actions of group members. We then introduce a stochastic optimization algorithm which attempts to minimize the number of labels required to assemble a structure. The algorithm operates by iteratively refining a worst-case solution, verifying at each step that the modified rules preserve all the constraints necessary for successful assembly.

II. Related Work

Our research was initially motivated by a desire to apply models for social insect nest construction to distributed robotic systems. In one such model, based on wasp nest construction and introduced by Bonabeau *et al.* [1], agents traverse a three-dimensional lattice, attaching cubic blocks together according to local assembly rules. The space of possible rules is explored by genetic algorithm in [2], with the primary goal of generating 'structured' architectures similar to those found in nature. The authors do not address the problem of building pre-specified shapes, however.

There have been significant recent efforts towards developing an algorithmic theory of self-assembly. In [4], Adleman

1-4244-0259-X/06/$20.00 ©2006 IEEE

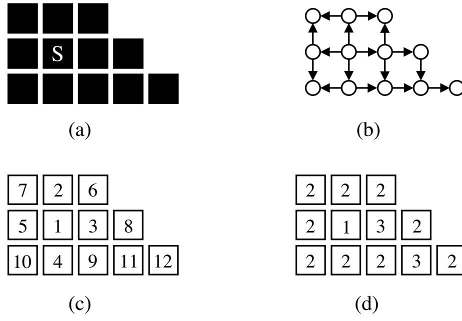

Fig. 1. Structure A from [3]. (a) Original structure; the seed is marked with an **S**. (b) Assembly graph generated by breadth-first traversal of A. (c) Worst-case labeling for A, $|\mathcal{L}| = 12$. (d) Optimized labeling for A, $|\mathcal{L}| = 3$.

suggests a model for one-dimensional self-assembly of square planar tiles and gives several associated complexity measures. Based on this work, Rothemund and Winfree introduce a two-dimensional *Tile Assembly Model* for self-assembled N × N squares in [5]. Each tile has a certain *type*, based on the *glue* values assigned to its four edges, and bonds to other tiles with compatible glues. The problem of finding the minimum number of tile types required to assemble an arbitrary planar structure is shown to be NP-complete in [6].

An alternative model for self-assembly and distributed robotic assembly based on graph grammars is presented by Klavins *et al.* [7]. Graph vertices represent parts, and an edge between two vertices indicates that the parts are attached. A graph grammar then defines a set of assembly rules which can be programmed into the system, by proper selection of the parts, to synthesize a desired structure. Ghrist and Lipsky extend graph grammars to tile assembly systems in [8].

The model most closely related to our own is defined by Jones and Matarić [3]. They describe a system in which unit-square assembly agents intelligently self-assemble into planar structures under rule-based control. Each agent is a finite state automaton that moves randomly on a two-dimensional grid. An agent is able to detect the states of neighboring agents in adjacent cells, and binds to those neighbors in response to certain state patterns. The agent then transitions to a new state, as defined by an internal rule set. This rule set is generated by an offline compiler which takes the goal structure as an input and produces the necessary assembly rules as an output.

No attempt is made in [3] to reduce number of unique states appearing in the assembly rules. Li and Zhang [9] address this issue by partitioning the goal structure into rectangular regions and generating rules independently for each region. Individual rectangles are encoded using the minimum number of states required. Partitioning is effective in many cases, although the algorithm does not take advantage of symmetry and does not provide improvement for 'degenerate' rectangles (those with a length or width of one unit).

Other assembly models use components which are more capable, but necessarily more complex, than those above. Arbuckle and Requicha [10] propose a model for *active* self-assembly, in which the agents are able to perform simple computations and send state messages to connected neighbors. Their approach allows for the assembly of partially-specified structures and of temporary scaffolds. Similarly, Werfel *et al.* [11] describe a heterogeneous system of blocks and mobile robots, where the blocks share structural information and communicate that information to nearby robots.

The literature also contains examples of distributed construction involving physical robots. Of particular relevance to our work is a study by Wawerla *et al.* [12] of the role of communication in a multi-robot construction task, where mobile robots assemble linear barriers composed of colored cylinders.

III. A Model for Distributed Assembly

In our distributed assembly model, homogeneous *assembly agents* move randomly and asynchronously on a two-dimensional grid of cells, transporting and attaching inert unit-square blocks together to form a desired *target structure* (or simply *target*). Every cell is either empty or completely filled by a single block. Our abstraction does not explicitly consider the spatial extent of the agents themselves; we assume that an agent is able to fit within one cell.

The position of a cell is represented by a coordinate pair (i, j) of integers, relative to an arbitrary origin $(0, 0)$. Each cell has four adjacent neighbors to the north, east, south and west. We select our coordinate axes such that the x axis increases to the east and the y axis increases to the north. Two cells at positions (m, n) and (i, j) are then adjacent if $|m - i| + |n - j| = 1$. The agents share a global sense of orientation and agree on the directions of adjacent cells.

Although all blocks have the same size and shape, each may be distinguished by an agent-assigned *label*. In a physical system, this label might be any feature that is identifiable at the scale of the agents, such as a color (macro-scale), inscription (micro-scale), or particular surface molecule (nano-scale). For the purposes of our model, a label is a value from the set $\mathbb{N}^+$ of positive integers. The label on a block cannot be changed once it has been assigned. We use the label '0' to denote an empty cell.

Assembly begins with a single *seed* block, which is fixed at the origin and assigned the label '1'. Growth of the structure occurs outwards from the seed. As an agent moves between empty cells, it compares the pattern of blocks in its local sensing neighborhood, consisting of the four adjacent cells,

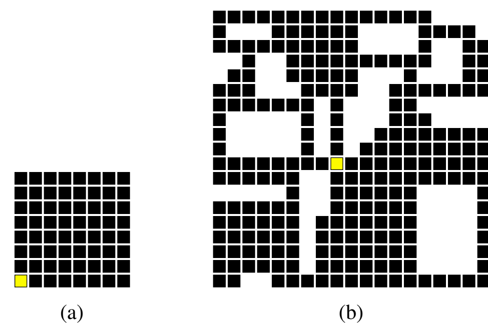

Fig. 2. (a) Structure B and (b) Structure C from [3]. The seeds are marked in yellow (light gray).

with entries in an internal lookup table of *assembly rules*. Each rule specifies a *binding configuration* that triggers an assembly action, and a *resultant* label. The binding configuration enumerates *a*) which adjacent cells must be occupied, and *b*) what labels must appear on blocks in the occupied cells. At least one cell in a binding configuration must be occupied, and up to three cells may be occupied. Once a binding configuration has been identified, an agent *applies* the corresponding rule by attaching a block to the structure at the agent's current position. This block is assigned the resultant label defined by the activated rule. Attachment is irreversible – assembly actions are never undone. We assume that an agent is able to attach the appropriate labeled block whenever a matching binding configuration is found.

Together, the assembly rules form an *assembly rule set*, and encode the information required to build the target structure. The rule set is identical for all the agents, allowing one agent to complete the assembly task alone if necessary.

Formally, we define a structure $\mathcal{T}$ as a set of coordinate pairs of occupied cells, and a *label set* $\mathcal{L}$ as $\mathcal{L} \subset \mathbb{N}^+$, with $|\mathcal{L}|$ finite. We model the assembly of *connected* structures only, in which every block is adjacent to a least one other block. If two cells (m,n) and (i,j) are adjacent, we describe the position of (m,n) with respect to (i,j) by writing $(m,n)^d_{(i,j)}$, where $d \in \{n,e,s,w\}$ is an abbreviation for the cardinal direction, north, east, south or west, of (m,n) relative to (i,j) on the grid. Two structures $\mathcal{T}_1$ and $\mathcal{T}_2$ are said to be *isomorphic* iff there is a translation vector v such that adding v to every element of $\mathcal{T}_1$ results in $\mathcal{T}_2$.

The assembly process yields a labeled structure $(\mathcal{T},\lambda)$, where $\mathcal{T}$ is a structure and λ is a surjection $\lambda : \mathcal{T} \to \mathcal{L}$. We will normally represent a labeled structure as a set of ((*coordinates*), *label*) pairs, or by a symbol $\bar{\mathcal{S}}$ with an over-bar. For a grid M, we use the notation $M(i,j)$ for the label on the block in the cell at position (i,j) if the cell is occupied, or 0 if the cell is empty. Likewise, since a labeled structure occupies a subset $\bar{\mathcal{S}} \subset M$ of the cells on a grid M, we also use the notation $\bar{\mathcal{S}}(i,j)$ for the label on the block in cell (i,j) which is part of $\bar{\mathcal{S}}$. A labeled structure is *complete* when it contains exactly the same set of occupied cells as the target.

An assembly rule is a five-tuple $(\rho_n,\rho_e,\rho_s,\rho_w,\gamma)$, where $\rho_n,\rho_e,\rho_s,\rho_w \in \mathcal{L} \cup \{0\}$ define the binding configuration and $\gamma \in \mathcal{L}$ is the resultant label. A rule may be applied at cell (i,j) on grid M iff $M(i,j) = 0 \wedge \rho_n = M(i,j+1) \wedge \rho_e = M(i+1,j) \wedge \rho_s = M(i,j-1) \wedge \rho_w = M(i-1,j)$.

Our assembly model is similar to the model in [3], except for the following differences. We disallow *'don't care'* entries within a binding configuration: the configuration for each assembly rule must be fully specified, and must exactly match the configuration in an agent's sensing neighborhood for a rule to be activated. Also, we do not use a block's position (or sector) relative to the seed to define which entries in a binding configuration must be non-zero. These changes allow us to assemble a larger class of structures than in [3].

Algorithm 1 Consistent Rule Set Decision

Input: Target $\mathcal{T}$, rule set $\mathcal{R}$ and assembly ordering $(\mathcal{T},\prec)$
Output: *True* if $\mathcal{R}$ is consistent for $\mathcal{T}$, *False* otherwise

```
1: if R contains conflicting rules then
2:    return False
3: end if
4: S̄ ← {((0,0),1)}
5: B ← {empty cells adjacent to (0,0)}
6: while B ≠ ∅ do
7:    B* ← ∅
8:    for each cell (i,j) ∈ B do
9:       if ∄ r ∈ R | r is potentially applicable at (i,j) then
10:         continue
11:      else if (i,j) ∉ T then
12:         return False
13:      else if ∃ r ∈ R | applying r would violate ≺ then
14:         return False
15:      end if
16:      r ← single rule from R applicable at (i,j)
17:      S̄ ← S̄ ∪ {((i,j), RESULTANT(r))}
18:      B* ← B* ∪ {empty cells adjacent to (i,j)}
19:   end for
20:   B ← B*
21: end while
22: if |S̄| < |T| then
23:    return False
24: end if
25: return True
```

IV. Spatiotemporal Coordination

Because assembly agents move randomly, the exact sequence in which blocks are attached to a growing structure cannot usually be determined in advance. However, the geometry of the structure imposes certain spatiotemporal ordering constraints on this sequence. For example, if we consider a solid 3×3 square target with the seed positioned in the southwest corner, the center block must be attached before the outer border of the structure is closed. Otherwise, agents will not be able to access the interior, making completion of the structure impossible.

We encode the spatiotemporal constraints in a rule set by first choosing an attachment order for each pair of adjacent blocks in the target, creating a partial ordering over all the blocks. This partial ordering defines *valid* sequences of assembly steps that always produce the desired structure. We call the partial ordering $(\mathcal{T}, \prec)$ an *assembly ordering*, where $\mathcal{T}$ is the target and $\prec$ is a transitive antisymmetric *predecessor* relation over $\mathcal{T}$. The predecessor relation defines which blocks must be in place before a new block may be attached at a specific position. Every block except for the seed has at least one predecessor.

An assembly ordering can be represented as a directed acyclic graph, called an *assembly graph*. There is a vertex in the graph for each block in the target, and a directed edge from vertex *i* to vertex *j* if the corresponding block b_i is adjacent

Algorithm 2 Rule Set from Labeled Structure

Input: Labeled structure $\bar{\mathcal{S}}$ and assembly ordering $(\mathcal{T}, \prec)$
Output: Rule set $\mathcal{R}$
1: $\mathcal{R} \leftarrow \emptyset$
2: **for** each cell $(i,j) \in \mathcal{T}$ **do**
3: **for** cell $(m,n)^d_{(i,j)}$ adjacent to (i,j), $d \in \{n,e,s,w\}$ **do**
4: **if** $(m,n) \in \mathcal{T}$ and $(m,n) \prec (i,j)$ **then**
5: $\rho_d \leftarrow \bar{\mathcal{S}}(m,n)$
6: **else**
7: $\rho_d \leftarrow 0$
8: **end if**
9: **end for**
10: $\mathcal{R} \leftarrow \mathcal{R} \cup (\rho_n, \rho_e, \rho_s, \rho_w, \bar{\mathcal{S}}(i,j))$
11: **end for**
12: **return** $\mathcal{R}$

to block b_j and b_i is a predecessor of b_j. A sink vertex in the graph is called a *terminal vertex*, because it represents a block to which no additional blocks will be attached. There is a single source vertex in the graph corresponding to the seed. An example assembly graph for the target structure in Figure 1*a* is shown in Figure 1*b*.

Given a rule set $\mathcal{R}$ and a target structure $\mathcal{T}$, an immediate question is whether $\mathcal{R}$ assembles $\mathcal{T}$ only, and no other structures. If the rules in $\mathcal{R}$ can be applied on a grid containing the seed, and the assembly process terminates with structure $\mathcal{T}$, for any possible sequence of actions by the agents, then $\mathcal{R}$ is said to be *consistent* [1] for $\mathcal{T}$. Consistency implies that $\mathcal{R}$ must not contain:

- any *conflicting* rules, which, for the same binding configuration, have different resultant labels.
- a rule which would *violate* the assembly ordering. We say the assembly ordering is violated if the rule allows a block to be attached before one or more of the predecessor blocks are in place.
- a rule which would add a block not in the target.

The first requirement above ensures that no ambiguous rules appear in the rule set, the second that the assembly process remains coordinated, and the third that only correct blocks are added to the growing structure. It is important to emphasize that there are many rule sets which *can* produced the desired target, if agents happen by chance to "do the right thing" and assemble blocks in the correct sequence. We are concerned with generating constrained rule sets that guarantee assembly of the target – the space of rule sets that have this guarantee is considerably smaller that the space of rule sets that are statistically *able* to produce the desired structure with some non-zero probability.

Algorithm 1 can be used to determine if a rule set $\mathcal{R}$ is consistent for a particular target $\mathcal{T}$. Before discussing the algorithm, we make an additional note regarding assembly rules. A rule r may *potentially* be applied at a position (i,j)

[1]The term *consistent* was originally introduced by Jones and Matarić [3].

Algorithm 3 Stochastic Rule Set Contraction

Input: Target structure $\mathcal{T}$
Output: Optimized, consistent rule set $\mathcal{R}$
1: $(\mathcal{T}, \prec) \leftarrow$ assembly ordering generated by $\mathrm{BFS}(\mathcal{T})$
2: $\mathcal{R} \leftarrow$ worst-case, $|\mathcal{T}|$-label rule set for $\mathcal{T}$
3: $\bar{\mathcal{S}} \leftarrow$ labeled structure produced by $\mathcal{R}$
4: **while** termination condition not met **do**
5: Randomly select non-seed cell $(i,j) \in \mathcal{T}$
6: **with** probability $1-\alpha$ **do**
7: $\bar{\mathcal{S}}(i,j) \leftarrow$ random label from interval $[1, \bar{\mathcal{S}}(i,j))$
8: **else**
9: $\bar{\mathcal{S}}(i,j) \leftarrow$ random label from interval $(\bar{\mathcal{S}}(i,j), |\mathcal{T}|]$
10: **end with**
11: $\mathcal{R}^* \leftarrow$ rule set generated from $(\bar{\mathcal{S}}, (\mathcal{T}, \prec))$ by Algorithm 2
12: **if** two rules in $\mathcal{R}^*$ conflict **then**
13: Prune rule with larger resultant label from $\mathcal{R}^*$
14: **end if**
15: $\bar{\mathcal{S}}^* \leftarrow$ labeled structure produced by $\mathcal{R}^*$
16: **if** $\mathcal{R}^*$ is consistent for $\mathcal{T}$ **then**
17: Prune any unused rules from $\mathcal{R}^*$
18: $\mathcal{R} \leftarrow \mathcal{R}^*$
19: $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}}^*$
20: **end if**
21: **end while**
22: **return** $\mathcal{R}$

if there is a valid sequence of block attachments that allows the binding configuration for r to appear at (i,j). This is a local test that accounts for all possible ways in which the structure could have grown such that position (i,j) lies on the boundary – that is, we evaluate whether a rule could be applied at (i,j) if any existing, adjacent predecessors were absent. For example, two rules, $(0,0,0,4,8)$ and $(0,0,3,4,8)$, may both potentially be applied at the same cell but at different stages of the assembly process, depending on whether the adjacent block with label '3' has been attached. An informal outline of algorithm correctness follows.

Algorithm 1 verifies initially that the binding configuration for each rule in $\mathcal{R}$ is unique, so we now must show that the algorithm returns *True* iff the assembly process terminates with the desired target structure $\mathcal{T}$.

Assume that $\mathcal{R}$ is not consistent (*inconsistent*) for $\mathcal{T}$, and let $\bar{\mathcal{V}}$ be another labeled structure produced by $\mathcal{R}$. Choose a sequence in which blocks are added to $\bar{\mathcal{V}}$, and let the cell (i,j) be the first position in the sequence where $(i,j) \notin \mathcal{T}$. If at any point a block can be added at (i,j) such that $(i,j) \notin \mathcal{T}$, the algorithm will discover a witness for the inconsistency (line 11), because every rule is tried at each boundary cell on each iteration.

Likewise, let (i,j) be the first position where the assembly ordering is violated. To violate the ordering, a rule must be potentially applicable at (i,j), for some binding configuration where one or more predecessors are missing. The algorithm will also posses a witness for inconsistency in this case,

TABLE I

COMPARISON OF RANDOMIZED CONTRACTION WITH TRANSITION RULE SET COMPILER (JONES AND MATARIĆ) AND RECTANGULAR PARTITIONING (LI AND ZHANG) FOR STRUCTURES A, B AND C GIVEN IN [3].

	Transition Rule Set		Rectangular Partitioning		Stochastic Contraction		
Structure	Labels	Steps	Labels	Steps	Labels	Steps	Iterations
A (12 blocks)	**5**	4	**9**	4	**3**	4	97
B (64 blocks)	**26**	14	**16**	14	**14**	14	1344
C (245 blocks)	**165**	18	**131**	18	**62**	18	31424

since, when a rule is tried at a boundary cell, all possible combinations of adjacent predecessors are considered (line 13).

Conversely, the algorithm only adds a block to the growing structure $\bar{\mathcal{S}}$ at position (i,j) when exactly one rule r is applicable (line 16), and only when the binding configuration for r includes all adjacent predecessors. These are precisely the requirements for coordinated assembly defined previously.

The main for loop terminates when no additional rules can be applied at any cells on the boundary of $\bar{\mathcal{S}}$. If, at this point, $\bar{\mathcal{S}}$ contains fewer blocks than $\mathcal{T}$, then the rule set fails to assemble the entire target and the algorithm returns *False* (line 22). Otherwise, $\mathcal{R}$ is consistent for $\mathcal{T}$ and the algorithm returns *True*.

To determine the time complexity, we first note that we can check for conflicting rules in $O(|\mathcal{R}|^2)$ time by pairwise comparison. On each iteration of the for loop we try $|\mathcal{R}|$ rules at an empty cell (i,j). When a rule is applied at (i,j), no more than three adjacent cells are added to $\mathcal{B}$ on the next iteration, while (i,j) is removed. The loop runs once for every block added to $\bar{\mathcal{S}}$, for time at most $O(|\mathcal{R}||\mathcal{T}|)$. This gives an overall time complexity of $O(|\mathcal{R}|^2 + |\mathcal{R}||\mathcal{T}|)$.

In this paper, we consider only the class of structures for which an assembly ordering can be generated by a breadth-first traversal of the blocks in the target, starting at the seed. This is a large class, however it excludes structures containing certain types of holes with non-convex interior boundaries. For some structures, there is no valid assembly ordering using the assembly model we have defined; we discuss these cases in Section VII.

V. GENERATING CONSISTENT, OPTIMIZED RULE SETS

Given a valid assembly ordering, we can always generate a consistent rule set for a target $\mathcal{T}$ in the following way. We begin by assigning a unique label to every block in $\mathcal{T}$, creating a labeled structure $\bar{\mathcal{S}}$. Then, as described by Algorithm 2, we step through the assembly ordering block by block, adding rules sequentially to the rule set. Each rule has a binding configuration defined by the adjacent predecessors (with other adjacent cells empty) and uses the resultant label already assigned to the block. This procedure is the reverse of Algorithm 1 – we extract, from a labeled structure, the series of rule applications that must have been used to produce it.

In general, the rule set returned by Algorithm 2 may contain conflicts and may not be consistent for $\mathcal{T}$, as the algorithm has no knowledge of how the labels were initially assigned. However, if the label on each block in $\bar{\mathcal{S}}$ is unique, and we follow the assembly ordering, consistency is *a priori* guaranteed . There is only one rule that can possibly be applied at any position and at any stage of the assembly process, because each resultant label appears only once in the complete structure.

The downside of this approach is that $|\mathcal{T}|$ labels are required for a $|\mathcal{T}|$-block target structure. We call such a $|\mathcal{T}|$-label rule set the *worst-case* rule set for $\mathcal{T}$, and use this worst-case solution as a starting point for our optimization algorithm.

A. Optimization by Stochastic Contraction

We can now formulate the following combinatorial optimization problem: for a target $\mathcal{T}$, find a rule set $\mathcal{R}$ which is consistent for $\mathcal{T}$ and uses the minimum number of labels possible. We do not know of a polynomial time algorithm to solve this problem, however the related Minimum Tile Set problem is NP-complete [6].

Our optimization technique, described by Algorithm 3, attempts to iteratively reduce the number of labels appearing in the rule set for $\mathcal{T}$. Each successful reduction is termed a *contraction*. Starting with a valid assembly ordering, we generate a consistent, worst-case rule set $\mathcal{R}$ and the associated labeled structure $\bar{\mathcal{S}}$. Then, at each iteration, we randomly select a block in $\bar{\mathcal{S}}$, change the label on this block, and generate a new rule set $\mathcal{R}^*$. If $\mathcal{R}^*$ is consistent for $\mathcal{T}$ the change is accepted, otherwise the change is rejected. This process continues until a user-defined termination condition is met, for example when a certain number of iterations have been completed.

A single parameter, α, determines how the label on a block is updated: with probability 1 - α we select a new label which

TABLE II

OPTIMIZED RULE SET FOR STRUCTURE IN FIGURE 1a. LABEL SET IS $\mathcal{L} = \{1,2,3\}$.

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

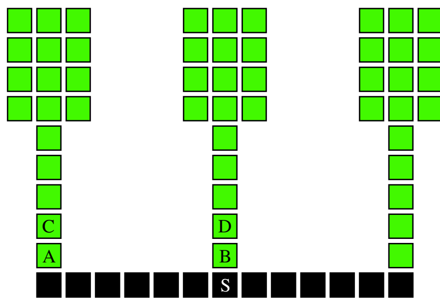

Fig. 3. Structure D, containing three isomorphic substructures, colored green (gray). The seed is marked with an **S**.

is smaller than the existing value; otherwise, we select a new label which is larger, up to the bound $|T|$. Typical values for α in our experiments lie in the range 0.005 to 0.01. Over a series of successful iterations, the bias towards smaller label values results in a decrease in the number of unique labels in $\mathcal{R}$, with some labels appearing on multiple blocks in $\bar{\mathcal{S}}$. The acceptance, with low probability, of larger label values provides an opportunity to reverse changes which caused the algorithm to reach a local, rather than the global, minimum label set size. As such, our approach resembles simulated annealing, except that the value of α remains constant over all iterations.

The complexity of Algorithm 3 is dominated by time required to verify that the updated rule set is consistent, and thus a single iteration can be implemented to run in $O(|\mathcal{R}||\mathcal{T}|)$ time using Algorithm 1, since $|\mathcal{R}|$ is always less than $|\mathcal{T}|$.

B. Conflicts and Isomorphisms

In some cases, changing the label on a block results in two conflicting rules appearing in the updated rule set $\mathcal{R}^*$. If this occurs, one of the rules must be removed. We choose to prune the rule with the larger resultant label.

Pruning will frequently introduce a discontinuity in the rule set, preventing a portion of the structure from being completed because a required rule is missing. However, for a target containing two or more isomorphic substructures sharing identical (oriented) assembly graphs, the worst-case rule set includes disjoint subsets of rules that assemble the same shape using different labels. Only one of these disjoint subsets is required. If the discontinuity due to pruning occurs at a position within an isomorphism, we can sometimes eliminate the redundant subsets of rules (and any unique labels they contain) immediately. This is possible when the label change allows one subset of rules to assemble two or more (isomorphic) substructures, as illustrated by the following example.

Consider structure D, shown in Figure 3, which contains three isomorphic substructures (the upright 'T' shapes). Blocks A, B, C and D in the structure will initially have different labels. Assume that, during an iteration of the optimization algorithm, the label on block B is randomly changed so that it is the same as the label on block A, while the labels on blocks C and D remain the same. The updated rule set $\mathcal{R}^*$, generated by Algorithm 2, will contain two conflicting rules for the (A, C) and (B, D) block pairs. If we assume also that the label on block D is greater than the label on block C, Algorithm 3 will prune the second rule. This breaks the chain of applications that led to the attachment of block D. However, when the algorithm applies $\mathcal{R}^*$, it immediately discovers that the subset of rules used to assemble the left substructure, including blocks A and C, *can also be used to assemble the central substructure*, including blocks B and D. The redundant rules for the central substructure are removed before the start of the next iteration.

TABLE III

OPTIMIZATION RESULTS FOR STRUCTURES D AND E.

Structure	Labels	Steps	Iterations
D (64 blocks)	**17**	16	1704
E (1045 blocks)	**149**	121	38618

VI. EXPERIMENTS

We have tested our optimization algorithm on a wide variety of structures, including the three examples given by Jones and Matarić in [3] and examined in [9]. For these structures, our assembly model and the models presented in [3] and [9] are equivalent, and hence the performance of the corresponding algorithms may be compared directly, as is done in Table I. Column 8 of the table gives the number of iterations required to achieve our optimized result, using α = 0.005.

In all three cases, the stochastic contraction algorithm generates rule sets which use fewer labels than the competing algorithms. For structures A (Figure 1*a*) and B (Figure 2*a*), it is possible to verify that our algorithm generates rule sets which are optimal by our criterion, using the minimum number of labels necessary to assemble these structures. We list the optimized rule set for structure A in Table II. For the largest and most complex structure, C (Figure 2*b*), our algorithm reduces the number of labels required by 62% and 52%, compared with [3] and [9] respectively.

Table I also lists the number of assembly steps for each structure. This is the number of steps required, starting with the seed only, to assemble the entire target, if all binding sites on the boundary of the growing structure are filled synchronously at every step. The metric, introduced in [3], corresponds to the length of the longest path in the assembly graph, which

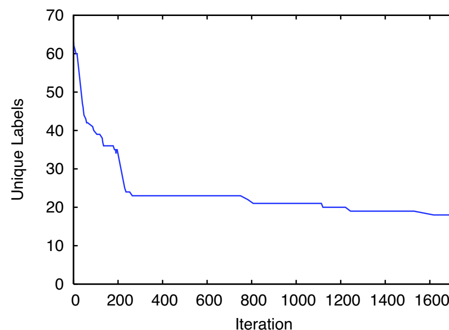

Fig. 4. Unique labels versus optimization iteration for Structure D, shown in Figure 3.

is independent of the number of labels used. Entries in each table column are identical because [3] and [9] implicitly use the same graph as our algorithm for structures A, B and C.

We mention two additional results here, shown in Table III. Figure 3, discussed in Section V-B, is an example of a structure containing several isomorphic substructures. The plot in Figure 4 gives the number of unique labels as a function of optimization iteration for this structure. Large decreases in the label count on iterations 41 (12 labels eliminated) and 231 (10 labels eliminated) are due to the algorithm's discovery of the isomorphisms and the consequent rule and label pruning. Figure 5 is a 1045-block structure that can be assembled using an optimized rule set containing only 149 labels. This is an 86% improvement over the worst-case.

VII. Discussion

The results in Tables I and III demonstrate that our optimization algorithm is able to reduce the number of labels required to assemble a variety of structures. When compared with [3] and [9], the relative performance of the algorithm improves as the complexity of the structure, defined by its size and geometric description, increases. Further, the algorithm works for a larger class of planar structures than [9].

The use of local sensing in our assembly model does have a drawback: it is not possible to generate consistent rule sets for structures such as the one shown in Figure 6 using a single seed. Blocks A and B in the figure correspond to terminal vertices in the assembly graph, and therefore represent the end of the flow of information about the state of the assembly process. Neither A nor B is a predecessor of any other block, and so assembly of the remainder of the structure is not impeded by their absence. There is no assembly ordering that enforces the constraints necessary to ensure that the structure border remains open until both A and B are attached.

VIII. Conclusions and Future Work

We have presented a simple model for distributed assembly and a stochastic optimization algorithm which reduces the number of labels that agents must recognize in order to complete an assembly task. The algorithm is scalable and significantly outperforms existing alternatives. We expect the model and optimization approach to be useful for distributed assembly tasks in collective robotic systems where sensing capabilities are limited.

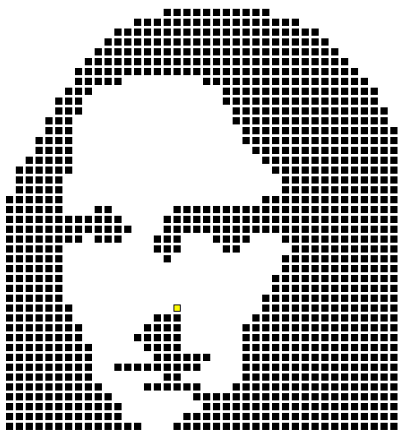

Fig. 5. Structure E, Leonardo da Vinci's Mona Lisa, containing 1045 blocks. The seed is marked in yellow (light gray).

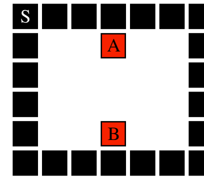

Fig. 6. A connected structure for which there is no consistent rule set using our single-seed model.

There are several directions for future work. Extending the assembly model to three dimensions is straightforward, and we are currently testing our algorithm's performance in this larger rule space. We also plan to investigate the use of heuristics to guide where in a structure optimization efforts should be concentrated. Finally, we would like to formally characterize classes of structures for which the assembly ordering constraints can be partially relaxed, while still ensuring that a deterministic result is produced.

References

[1] E. Bonabeau, G. Théraulaz, E. Arpin, and E. Sardet, "The building behavior of lattice swarms," in *Proc. Fourth Int'l Conf. Artificial Life*. Cambridge, USA: MIT Press, 1994, pp. 307–312.

[2] E. Bonabeau, M. Dorigo, and G. Théraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[3] 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, Sep 2003, pp. 721–726.

[4] L. Adleman, "Towards a mathematical theory of self-assembly," Department of Computer Science, University of Southern California, Los Angeles, USA, Tech. Rep. 00-722, 2000.

[5] 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, 2000, pp. 459–468.

[6] 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, 2002, pp. 23–32.

[7] E. Klavins, R. Ghrist, and D. Lipsky, "Graph grammars for self assembling robotic systems," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '04)*, New Orleans, USA, Apr 2004, pp. 5293–5300.

[8] 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.

[9] G. Li and H. Zhang, "A rectangular partition algorithm for planar self-assembly," in *Proc. IEEE Int'l Conf. Intelligent Robots and Systems (IROS '05)*, Edmonton, Canada, Aug 2005, pp. 2324–2329.

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

[11] J. Werfel, Y. Bar-Yam, and R. Nagpal, "Building patterned structures with robot swarms," in *Proc. Nineteenth Int'l Joint Conf. on Artificial Intelligence (IJCAI '05)*, Edinburgh, Scotland, Aug 2005, pp. 1495–1502.

[12] J. Wawerla, G. S. Sukhatme, and M. J. Matarić, "Collective construction with multiple robots," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS '02)*, Lausanne, Switzerland, Oct 2002, pp. 2696–2701.

[13] 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, 2001, pp. 740–748.