Concurrent metamorphosis of hexagonal robot chains into simple connected configurations

Jennifer E. Walter, Jennifer L. Welch, and Nancy M. Amato

Abstract— **The problem addressed is the distributed reconfiguration of a metamorphic robotic system composed of an arbitrary number of two dimensional hexagonal robots (modules) from specific initial to specific goal configurations. The initial configuration considered is a straight chain of robotic modules, while the goal configurations considered satisfy a more general "admissibility" condition. A centralized algorithm is described for determining whether an arbitrary goal configuration is admissible. We prove this algorithm correctly identifies admissible goal configurations and finds a "substrate path" within the goal configuration along which the modules can move to reach their positions in the goal. A second result of the paper is a distributed algorithm for reconfiguring a straight chain into an admissible goal configuration. Different heuristics are proposed to improve the performance of the reconfiguration algorithm and simulation results demonstrate the use of these heuristics.**

Keywords—**Metamorphic robots, distributed reconfiguration**

I. INTRODUCTION

A topic of recent interest in the field of robotics is the development of motion planning algorithms for robotic systems composed of a set of robots (modules) that change their position relative to one another, thereby reshaping the system. A robotic system that changes its shape due to individual robotic motion has been called *self-organizing* [6] or *self-reconfigurable* [10].

A self-reconfigurable robotic system is a collection of independently controlled, mobile robots, each of which has the ability to connect, disconnect, and move around adjacent robots. *Metamorphic* robotic systems [3], a subset of self-reconfigurable systems, are further limited by requiring each module to be identical in structure, motion constraints, and computing capabilities. Typically, the modules have a regular symmetry so that they can be packed densely, i.e., packed so that gaps are minimized between adjacent modules in a configuration that densely packs the plane. In these systems, robots achieve locomotion by moving over a substrate composed of one or more other robots. The mechanics of locomotion depends on the hardware and can include module deformation to crawl over neighboring modules [5], [16] or to expand and contract to slide over surfaces [17]. Alternatively, moving robots may be constrained to rigidly maintain their original shape, requiring them to roll over or lift themselves around neighboring robots or other surfaces [8], [12], [13], [14], [18], [19], [25], [26].

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as bridge building, satellite recovery, or tumor excision [16]. The complete interchangeability of the robots provides a high degree of system fault tolerance. Also, self-reconfiguring robotic systems are potentially useful in environments that are not amenable to direct human observation and control (e.g., interplanetary space, undersea depths).

The motion planning problem for a metamorphic robotic system is to determine a sequence of robot motions required to go from a given initial configuration (I) to a desired goal configuration (G).

Many developers of self-reconfigurable robotic systems, such as [4], [5], [6], [8], [10], [13], [14], [16], [17], [18], [24], and [25], have devised motion planning strategies specific to the hardware constraints of their prototype robots. Most of the existing motion planning strategies rely on centralized algorithms to plan and supervise the motion of the system components [1], [5], [10], [16], [17], [24]. Others use distributed approaches which rely on heuristic approximations and require communication between robots in each step of the reconfiguration process [11], [12], [14], [18], [25], [26].

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian [5]. We consider a distributed reconfiguration, given the assumption of initial global knowledge of G. For the purposes of this paper, we use the terms *concurrent*, *parallel*, and *distributed* synonymously. Our reconfiguration algorithm is distributed in the sense that modules are independent and move individually using only local information. During the reconfiguration, the modules move simultaneously, synchonized to a global clock. Our distributed approach offers the benefits of localized decision making and the potential for greater system fault tolerance. Additionally, our strategy requires less communication between modules than other approaches. We have previously applied this approach to the problem of reconfiguring a straight chain to an intersecting straight chain [23].

In this paper we address the problem of distributed reconfiguration from a straight chain of robots to goal configurations that satisfy a more general "admissibility" condition. A centralized algorithm is described for determining whether an arbitrary goal configuration is admissible, and if so, for finding a substrate path along which the robotic

The work of N. Amato and J. Walter was supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACI-0113971, CCR-0113974, EIA-9810937, EIA-0079874, and by the Texas Higher Education Coordinating Board grant ARP-036327-017. The work of J. Walter was supported in part by Department of Education GAANN and GE Faculty of the Future fellowships.

J. Walter (contacting author) is with the Department of Computer Science, Vassar College, Poughkeepsie, NY 12604-0351. FAX: (845)437-7498, E-mail: walter@cs.vassar.edu, Phone: (845)485-1413.

J. Welch and N. Amato are with the Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. FAX: (979)847-8578 Welch: E-Mail: welch@cs.tamu.edu, Phone: (979)845-5076. Amato: E-Mail: amato@cs.tamu.edu, Phone: (979)862-2275.

modules can move to reach the goal configuration. Then we present a distributed algorithm for reconfiguring a straight chain into an admissible goal configuration, using the path found by the centralized algorithm. Different heuristics for choosing the path are proposed to improve the efficiency of the reconfiguration algorithm and the performance of these heuristics is explored through simulation.

II. RELATED WORK

A. Centralized Reconfiguration Algorithms

Centralized motion planning algorithms have been developed to reconfigure both two and three dimensional metamorphic robots of various forms. Our planner is different because it starts with a centralized algorithm that finds a collision and deadlock free substrate path bisecting the goal configuration (if such a path exists). Then, unlike the algorithms presented in this section, multiple modules running our distributed algorithm initiate movement concurrently to perform the reconfiguration.

A number of centralized reconfiguration algorithms have been developed for planar hexagonal robots. Chirikjian [5] and Pamecha [16] present centralized algorithms that use the distance between two configurations as a metric that allows the choice of a single module to move in each round. Other metrics for reconfiguration are presented in [2]. Upper and lower bounds on the number of moves for reconfiguration between general shapes are given by Chirikjian [5].

Nguyen et al. [15] present a reconfiguration algorithm for planar hexagonal robots and show that the absence of a single excluded class of initial configurations is sufficient to guarantee the feasibility of motion planning for a system composed of a single connected component.

Casal and Yim [1] present algorithms to reconfigure a system of hexagonal robots in a hierarchical fashion, where reconfiguration involves connectivity changes within and between substructures. Their paper concentrates on the decomposition algorithms and does not present algorithms for motion planning within substructures.

Centralized motion planners for cubic robots are presented in [8], [17] and [19]. The modules in [17] are deformable, expanding and contracting to slide over adjacent modules or some other substrate. In [8] and [19], the cubic modules are non-deformable and move as a result of being lifted or pivoting over other modules.

B. Distributed Reconfiguration Algorithms

A number of distributed algorithms have been proposed for reconfiguring systems of metamorphic robots. The similarities and differences between our algorithm and the algorithms described in this section are discussed in Section III.

A distributed approach is taken by Murata et al. to reconfigure a system of planar hexagonal modules [12], and a system of cubic modules [14]. In these algorithms, each module senses its own connection type and classifies itself by the number of modules it physically contacts. Modules use a formula that relates their connection type to the set of connection types in the goal configuration to quantify their *fitness* to move. Modules communicate with physical neighbors to ensure that only the modules that have fitness greater than the local fitness average move in the same time step, choosing a direction at random. Modules use random local motions to converge toward the goal configuration, a slow process that appears impractical for large configurations.

Hosokawa et al. [8] present a distributed algorithm to reconfigure a collection of cubic modules. Their modules use a uniform set of embedded rules to determine their behavior based on the local environment and inter-module communication.

A distributed approach to the reconfiguration of tetrahedral modules (i.e., Tetrobots, described in [7]) is presented in [11]. In this approach, each processor uses a dynamic subsystem model and communication between adjacent modules to compute kinematics, dynamics, and control input necessary to move one dedicated module to a new position.

A reconfiguration algorithm for rhombic dodecahedral modules is presented by Yim et al. [25]. In this algorithm, each module uses local information about its own state (the number and location of its current neighbors) and information about the state of its neighbors obtained through inter-module communication to heuristically choose moves that lower its distance to the goal configuration.

Several heuristic approximation algorithms for reconfiguration of rhombic dodecahedral robots are presented by Zhang et al. [26]. In their two phase approach, modules use neighbor-to-neighbor communication in the first phase to achieve a semi-global view of the initial configuration, using as many rounds as necessary to avoid violation of module motion constraints prior to each phase of movement.

III. Our Approach

This paper will examine distributed motion planning strategies for a planar metamorphic robotic system undergoing a reconfiguration from a straight chain to a goal configuration satisfying certain properties. In our algorithms, robots are identical, but act as independent agents, making decisions based on their current position and the sensory data obtained from physical contacts with adjacent robots.

Our purpose is to seek an understanding of the necessary building blocks for reconfiguration, starting with algorithms in which no messages need to be passed between participating robots during reconfiguration. Reconfiguration in certain scenarios, like the ones presented in this and our earlier papers [23] and [22], can be accomplished using algorithms that do not require any message passing. Therefore, our algorithms are more communication efficient than the distributed approaches of [8], [11], [12], [25] and [26]. Another contribution of our work is that our system model abstracts from specific hardware details about the robots.

In this paper, we consider planar hexagonal robots like those described by Chirikjian [3], using the same definition of lattice distance between robots in the plane. Our

proposed scheme uses a new classification of robot types based on connected edges similar to the classification used by Murata et al. [12] for connected vertices. In the algorithms presented in this paper, each robot independently determines whether it is in a movable state based on the cell it occupies in the plane, the locations of cells in the goal configuration, and on which sides it contacts neighbors. Robots move from cell to cell and modify their states as they change position. Since the robots know the coordinates of the goal cells, we show that each of them can independently choose a motion plan that avoids module collision.

Unlike the distributed algorithms presented in [8], [12], [25], and [26], our algorithm is deterministic, using a fixed number of rounds for a particular choice of substrate path and a particular goal configuration. Our distributed reconfiguration will be initiated only if the goal configuration satisfies particular admissibility conditions. Once started, our distributed algorithm does not require inter-module message passing to avoid violation of motion constraints, module collision, or deadlock, as do the algorithms of [26].

In this paper we also present precise conditions for admissible goal configurations based on the motion constraints of our robots. We present an algorithm that ensures these admissibility conditions and prove that this algorithm correctly identifies admissible goal configurations. The admissibility conditions presented in this paper differ from those presented by Rus and Vona [17] and Nguyen et al. [15]. In the first case, this difference is due to module shape and motion constraints, and, in the second case, the difference is due to assumptions on module motion, as will be explained in Section V.

In Section IV we describe the system assumptions and the problem definition. Section V contains a centralized algorithm that determines whether or not a given configuration is admissible. Section VI presents and analyzes a distributed algorithm for reconfiguring a straight chain to an admissible goal configuration. In Section VII we present simulation results comparing the performance of our algorithm using different heuristics. Section VIII provides a discussion of our results and future work.

IV. SYSTEM MODEL

A. Coordinate System

The plane is partitioned into equal-sized hexagonal cells and labeled using the coordinate system shown in Fig. 1, as in Chirikjian [3].

Given the coordinates of two cells, $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, we define the *lattice distance*, LD, between them as follows: Let $\Delta x = x_1 - x_2$ and $\Delta y = y_1 - y_2$. Then

$$LD(c_1, c_2) = \begin{cases} \max(|\Delta x|, |\Delta y|) & \text{if } \Delta x \cdot \Delta y < 0, \\ |\Delta x| + |\Delta y| & \text{otherwise.} \end{cases}$$

The lattice distance describes the minimum number of cells a module would need to move through to go from cell c_1 to cell c_2.

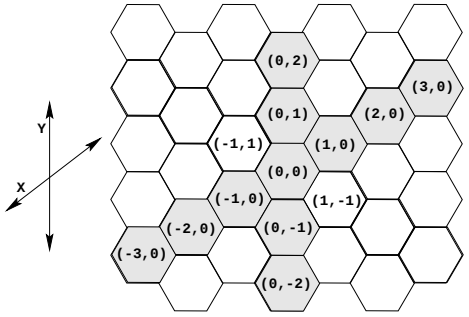

Fig. 1. Coordinates in a system of hexagonal cells.

B. Assumptions About the Modules

Our model provides an abstraction of the hardware features and the interface between the hardware and the application layer.

- Each module is identical in computing capability and runs the same program.
- Each module is a hexagon of the same size as the cells of the plane and always occupies exactly one of the cells.
- Each module knows at all times:
  - its location (the coordinates of the cell that it currently occupies),
  - its orientation (which edge is facing in which direction), and
  - which of its neighboring cells is occupied by another module.

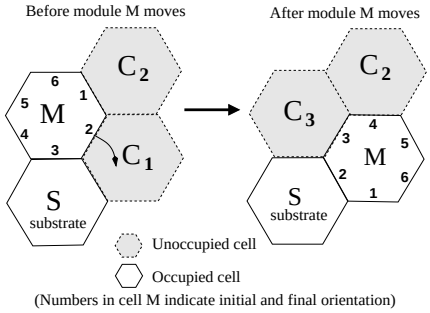

Fig. 2. Before (left) and after (right) module movement.

Modules move according to the following rules.

1. Modules move in lockstep rounds (synchronized by a global clock).
2. In a round, a module M is capable of moving to an adjacent cell, C_1, if and only if (see Fig. 2 for an example)
   (a) cell C_1 is currently empty,
   (b) module M has a neighbor S that does not move in the round (called the *substrate*) and S is also adjacent to cell C_1, and
   (c) the neighboring cell to M on the other side of C_1 from S, C_2, is empty.
3. Only one module tries to move into a particular cell in each round.
4. Modules cannot carry other modules, i.e., a module is only allowed to move itself.

If the algorithm does not ensure that each moving module has an immobile substrate, as specified in rule 2(b),

then the results of the round are unpredictable. Likewise, the results of the round are unpredictable if the algorithm does not ensure rule 3.

C. Problem Definition

We want a distributed algorithm that will cause the modules to move from an initial configuration, I, in the plane to a known goal configuration, G.

V. ADMISSIBLE CONFIGURATIONS

In this section we define admissible configurations and describe a centralized algorithm that tests whether a given configuration is admissible.

A. Definition of Admissible Configuration

To simplify the presentation of admissible goal configurations, assume I is a straight chain oriented north-south, no goal cell is to the west of I, and I and G intersect in the southernmost module of I and nowhere else. The number of modules in I and the number of cells in G is n. Figure 3 gives examples of orientations of I and G that satisfy these assumptions in which $n = 6$. The assumptions concerning the orientation of I and G can be made without loss of generality because, if I is a straight chain that is not oriented in this way, the algorithms presented in [23] for straight chain to straight chain reconfiguration can be used to reorient I in relation to G.

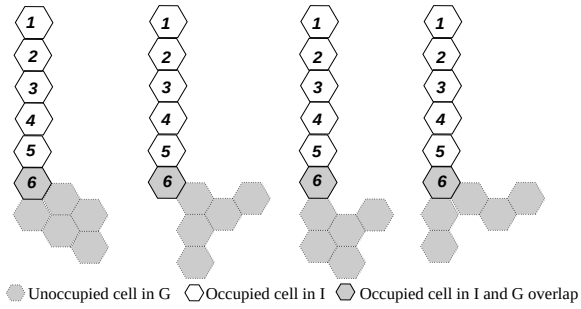

Fig. 3. Example orientations of I and G.

Let $G_1, G_2, \ldots, G_m$ be the columns of G from west to east such that each column is oriented north-south and each is composed of a contiguous chain of goal cells. Figure 4(a) shows how the columns of G are labeled and gives an example of a configuration of G in which each column is a contiguous chain of goal cells. Figure 4(b) gives an example of a configuration of G in which columns G_3 and G_5 are composed of non-contiguous chains of goal cells.

Let p be a contiguous sequence of distinct cells, $c_1, c_2, \ldots, c_k$. Then

Definition 1: p is a *substrate path* if
- p begins with the cells in I, from north to south,
- subsequent cells are all in G, and
- the last cell is in the easternmost column of G (G_m).

Definition 2: A *segment* of p is a contiguous subsequence of p of length ≥ 2. In a *south segment*, each cell is south of the previous cell and analogously for a *north segment*.

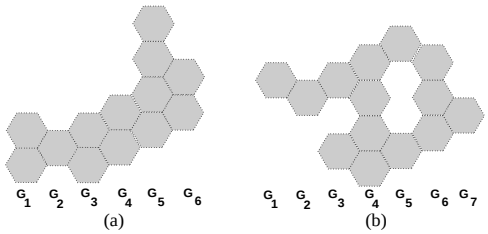

Fig. 4. Two configurations of G: (a) each column is composed of a contiguous chain of goal cells, and (b) columns G_3 and G_5 are composed of non-contiguous chains of goal cells.

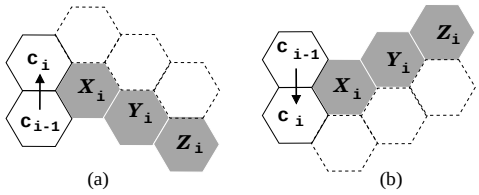

Fig. 5. Labels for north segment ending in c_i (a) and south segment ending in c_i (b) (cells that must not be goal cells are shaded).

Definition 3: p is an *admissible path* if
1. p is *east-monotone*, meaning that each cell in p is adjacent to the previous, *but not to the west* (i.e., consecutive higher numbered cells may not be on the northwest or southwest side of a given cell),
2. for each north segment of p ending with c_i,
   (a) cells X_i, Y_i, and Z_i, consecutive to the SE of c_i, are not goal cells (see Figure 5(a)) and
   (b) c_{i+1}, c_{i+2}, c_{i+3}, and c_{i+4} do not form any south segments,
   and
3. for each south segment of p ending with c_i,
   (a) cells X_i, Y_i, and Z_i, consecutive to the NE of c_i, are not goal cells (see Figure 5(b)) and
   (b) c_{i+1}, c_{i+2}, c_{i+3}, and c_{i+4} do not form any north segments.

In the remainder of this paper, north and south segments of p may be referred to as *vertical* segments when specific direction of the segment is not important. Conditions 2(a) and 3(a) of Definition 3 specify where a vertical edge may be added to p relative to goal cells in the three columns to the east. Conditions 2(b) and 3(b) say that any vertical segment of p must be separated from any vertical segment in the opposite direction and to the east by at least 3 columns.

Definition 4: G is an *admissible goal configuration* if there exists an admissible substrate path in G.

Intuitively, an admissible substrate path is a chain of goal cells whose surface allows the movement of modules without collision or deadlock, provided the choices of module rotation and delay are appropriate. That is, provided the motion planning algorithm allows for adequate space

between moving modules, there are no pockets or corners on the surface of the substrate path in which modules will become trapped.

The admissibility conditions for a substrate path are directly related to the degree of parallelism desired, i.e., how closely moving modules can be spaced. If moving modules are separated by only a single empty cell, they will become deadlocked in acute angle corners when running our algorithms [23]. However, acute angle intersections are very commonplace in configurations of hexagonal robots. Thus, we chose to make our algorithms applicable to a wide range of goal configurations by separating moving modules by two empty cells. *Our definition of admissibility is therefore based on configuration surfaces over which moving modules with two empty cells between them can move without becoming deadlocked.*

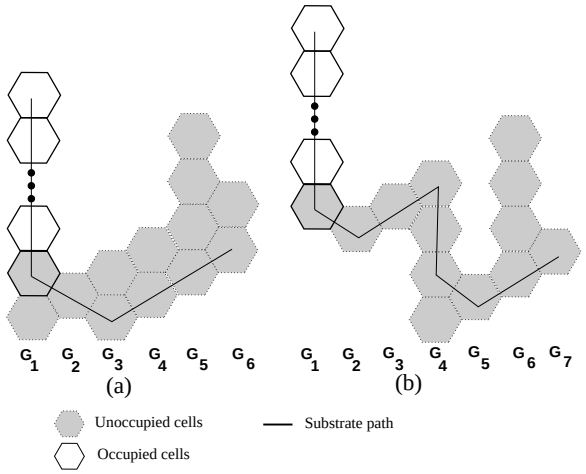

Fig. 6. Example admissible (a) and inadmissible (b) G.

Figure 6(a) depicts an example of an admissible configuration of G, where the line through I and G is an admissible substrate path. Figure 6(b) depicts a configuration of G with a substrate path that is inadmissible because the south segment in column G_4 violates Definition 3, part 3(a). Because any substrate path in the configuration of G shown in Figure 6(b) has to include a south segment in column G_4, there is no admissible substrate path for this configuration and therefore this configuration of G is inadmissible, by Definition 4.

Our definition of admissible classes of goal configurations differs from that presented by Rus and Vona [17] because the modules developed by these authors are cubic, with a different set of motion constraints and mode of locomotion. Even though our modules are two dimensional and hexagonal, like those of Nguyen et al. [15], our definition of admissible classes of goal configurations is different than theirs because our assumptions about module motion are different. Nguyen et al. assume that a module moves by rigid rotation around a vertex it shares with another module. Our motion constraints are similar to those presented by Chirikjian [3], where locomotion is accomplished by a combined rigid body rotation and shape transformation produced by changing joint angles.

B. Algorithms to Detect Admissible Configurations and Find Substrate Paths

As a first step in determining the admissibility of G, scan G in columns from north to south, northwest to southeast, and northeast to southwest, to determine if there exists an orientation in which each column G_i is contiguous. If there is no orientation in which each column G_i is contiguous, then G is not admissible.

Our procedure for finding an admissible substrate path in G proceeds by first constructing a directed graph H as follows:

- Label the columns of G as described in Section V-A, with the cells in each column G_i labeled $G_{i,1}$, $G_{i,2}$,..., from north to south. Then cell $G_{1,1}$ is also in I, but no other goal cells are in I.
- Represent each goal cell as a node in the graph H. Add an extra node to the graph in the cell directly north of cell $G_{1,1}$ and call this node $G_{1,0}$. Because of the orientation of I and G, the cells in I form a south segment of the substrate path which must be considered when determining if G is admissible. Initially there is an undirected edge between each pair of adjacent goal cells.
- The cells to the north, south, northeast, and southeast of $G_{i,j}$ are labeled $N_{i,j}$, $S_{i,j}$, $NE_{i,j}$, and $SE_{i,j}$, respectively (note that some of these cells might not be goal cells and thus are not represented in the graph).

- **Algorithm** DIRECT_EDGES
  - First, every node in column G_m is marked, as shown in Figure 7(a).
  - Each column west of column G_m (i.e., columns G_1 through G_{m-1}) is divided into three segments (labelled in Figure 7(a)): (N) the north segment of the column with no goal cells to the east (possibly empty), (C) the central segment of the column, consisting of cells that have goal cells to the east, and (S) the south segment of the column with no goal cells to the east (possibly empty).
  - For each column, G_{m-1} down to G_1
    1. Nodes in segment (C) are processed north to south. If a node in segment (C) has one or more marked neighbors to the east, it is marked and given a directed edge to each marked neighbor. *The only exceptions are when a NE edge would be directed toward a neighbor with an outgoing S edge or where a SE edge would be directed toward a neighbor with an outgoing N edge.* These exceptions ensure that no acute angle corners will be included in any substrate path.
    2. Nodes in segment (S) are processed north to south. Each node is marked and given a directed edge to its north neighbor if the north neighbor is marked and if the edge is a prefix of some admissible path.
    3. Nodes in segment (N) are processed south to north. Each node is marked and given a directed edge to its south neighbor if the south neighbor

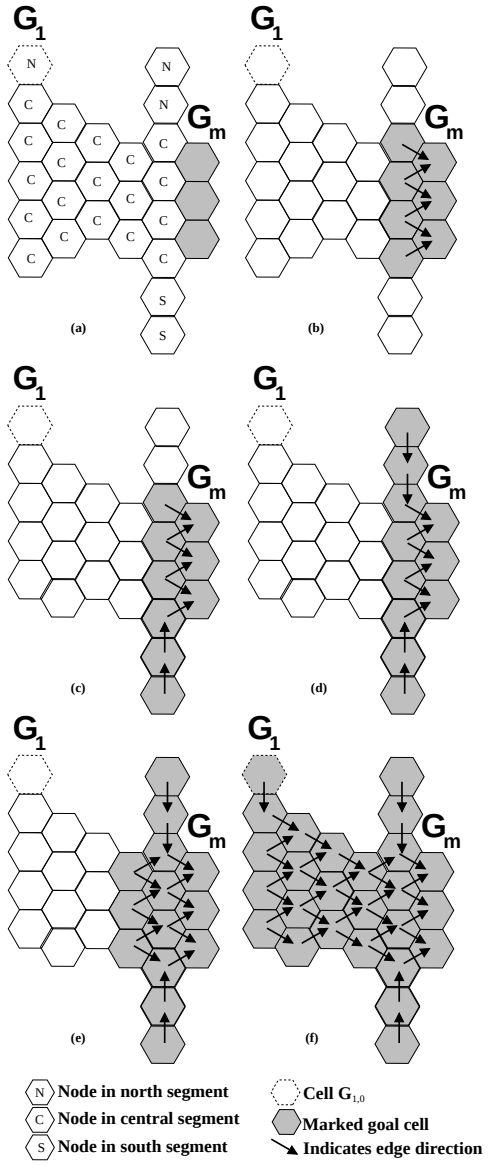

Fig. 7. Example execution of DIRECT_EDGES.

is marked and if the edge is a prefix of some admissible path.

Algorithm DIRECT_EDGES constructs H by directing edges in the undirected graph and marking the nodes that are determined to have an admissible path to a goal cell in the easternmost column. Figure 7(a)–(f) depicts an execution of DIRECT_EDGES, with execution progressing from (a) to (f). In part (a), column G_m is marked. In part (b), execution has progressed to column G_{m-1} and the cells in the central segment are marked and have edges directed toward marked cells in G_m. In part (c), nodes in the south segment of column G_{m-1} have been marked, from the top down, with edges directed toward marked cells in the central section. In part (d), nodes in the north segment of column G_{m-1} have been marked, from the bottom up, with edges directed toward marked cells in the central section. Execution has progressed to column G_{m-2} in part (e) and the cells in the central segment have been marked with edges directed toward marked cells in G_{m-1}. Part (f) shows the graph H after DIRECT_EDGES has finished execution.

Algorithm FIND_PATH, described below, is used to construct an admissible substrate path.

- **Algorithm** FIND_PATH:
  The input is the graph H after processing G with DIRECT_EDGES. If node $G_{1,0}$ is marked, add this node to the substrate path. Follow an outgoing edge from $G_{1,0}$ to $G_{1,1}$ if $G_{1,1}$ is marked, adding node $G_{1,1}$ to the substrate path.
  The remainder of the algorithm proceeds according to the following rules, stopping when any cell in column G_m is added to the substrate path:
  1. If a node has a directed edge to only one marked neighbor (either N, S, NE, or SE), then traverse the edge in that direction and add the edge to the substrate path.
     For every N vertical segment ending in goal cell $G_{i,j}$ $(1 \le i < m-3)$, the cell NE of cell $NE_{i,j}$ is unmarked if there is a S segment in column G_{i+3}. A symmetric action is taken for a S vertical segment.
  2. If a node has outgoing directed edges to two marked neighbors (NE and SE), choose one edge, traverse it, and add the edge to the substrate path (In Section VII-A, we discuss heuristics that can be used to choose either the NE or SE edge).

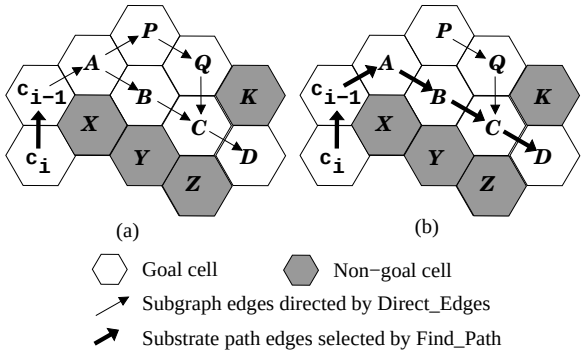

Fig. 8. Snapshots of FIND_PATH execution, (a) before and (b) after edge (c_i,c_{i-1}) is added to the substrate path.

If FIND_PATH adds a N or S edge in column G_i $(1 \le i < m-3)$, then a particular cell in the graph two columns to the east may be unmarked. For example, in Figure 8(a), edges (C,D), (Q,C), (P,Q), (B,C), (A,B), (A,P), $(c_{i-1}$,A) and (c_i,c_{i-1}) have been directed by algorithm DIRECT_EDGES prior to the execution of FIND_PATH. Suppose edge (c_i,c_{i-1}) has just been added to the substrate path in part (a). Then in part (b), cell P is unmarked so that the inadmissible substrate path containing edges (c_i, c_{i-1}), (c_{i-1},A), (A,P), (P,Q) and (Q,C) will not be chosen by FIND_PATH.

C. Analysis of Algorithms DIRECT_EDGES and FIND_PATH

The time required to check if the columns of G are contiguous is $O(n)$, since every cell in each column must be examined to look for "holes". The running time of the algorithm to find the graph H and to find an admissible substrate path is $O(n)$, since each node has a constant number of (undirected) neighbors.

We require that algorithm FIND_PATH returns a path that ends in column G_m if and only if G is admissible. We proceed with a sketch of the proof of correctness. For those readers interested, the full proof can be found in [20].

Theorem 1: If G is admissible, then FIND_PATH returns a path that ends in G_m.

We first show, in Lemma 1, that DIRECT_EDGES will mark cells on all admissible paths leading to any cell in column G_m.

Lemma 1: For every goal cell c, if there is an admissible path from c to a cell in G_m, then algorithm DIRECT_EDGES marks c.

The proof of Lemma 1 uses induction on the order in which DIRECT_EDGES scans goal cells, including a case analysis on all possible orientations of an outgoing edge at c, the next cell to be marked. We show that, if c is a prefix of an admissible path, then c must be marked by algorithm DIRECT_EDGES and an edge must be directed from c toward a marked goal cell to the NE, SE, N, or S.

Figure 9 depicts two snapshots of different subgraphs of G taken during the execution of DIRECT_EDGES in which a goal cell c_i with a marked neighbor to the north is not marked. In Figure 9(a), edges (C,D), (Q,C), (P,Q), (A,P), and (c_{i-1},A) have been directed consecutively by algorithm DIRECT_EDGES. When the edge from cell c_i to c_{i-1} is considered, it is not directed because including this edge in any substrate path would violate the definition of an admissible substrate path, Definition 3, part 2(b). In Figure 9(b), edge (c_i,c_{i-1}) is not directed because including this edge in any substrate path would violate Definition 3, part 2(a). If cell Y was a goal cell in Figure 9(b), including edge (c_i, c_{i-1}) in a substrate path would also violate Definition 3, part 2(a). If cell X was a goal cell in this figure, then c_i would have a NE, not a N edge, so edge (c_i, c_{i-1}) would not be included in any admissible substrate path (similar arguments with inverted snapshots can be made if the edge from c_i to c_{i-1} is a south edge). These are the only cases in which a goal cell with a marked neighbor is not marked, and in each case the goal cell c_i that is not marked is also not a prefix of any admissible substrate path.

Lemma 1 implies that if cell $G_{1,0}$ is marked, then it is the prefix of an admissible path that ends in column G_m. After DIRECT_EDGES finishes, if cell $G_{1,0}$ is marked, algorithm FIND_PATH builds a substrate path starting in cell $G_{1,0}$ and following directed edges until the path includes some cell in column G_m.

The remainder of the proof of Theorem 1 involves showing that at the time algorithm FIND_PATH adds vertical edges, thereby unmarking goal cells, the substrate path it has constructed to that point is still a prefix of an admissible path that ends in column G_m. To be rigorous, this proof requires us to show that when a north edge is added to the substrate path in column G_i, causing a goal cell to be unmarked by FIND_PATH in column G_{i+2}, the goal cells in columns G_i through G_{i+4} must be in a configuration like that shown in Figure 8(b). Therefore, an admissible substrate path will still exist after a north edge is added (a similar argument holds for a south edge).

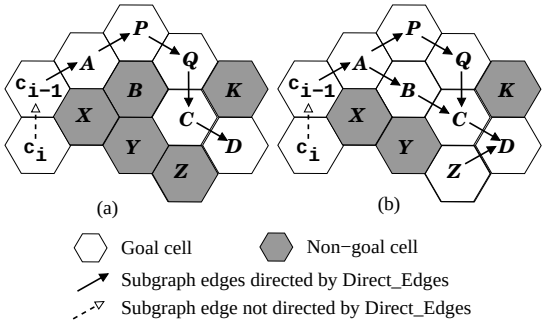

Fig. 9. Snapshots of DIRECT_EDGES execution depicting: (a) a situation in which a vertical edge is not added to H due to violation of Definition 3, part 2(b), and (b) a situation in which a vertical edge is not added to H due to violation of Definition 3, part 2(a). Initial undirected edges not shown.

Theorem 2: If algorithm FIND_PATH returns a path ending in column G_m, then G is admissible.

The proof of Theorem 2 uses induction on the order in which goal cells are added to the substrate path, including a detailed case analysis of the goal configuration to the west when each possible edge direction is added, to show that the substrate path satisfies the conditions in Definition 3.

Theorems 1 and 2 imply that algorithm FIND_PATH will return only an admissible substrate path and will find an admissible substrate path if one exists in G. In other words, the algorithms presented in this section will correctly identify admissible configurations of G.

VI. DISTRIBUTED RECONFIGURATION ALGORITHM

In this section, we present the distributed reconfiguration algorithm that performs the reconfiguration of I to G after an admissible substrate path is found using the algorithms in the previous section. This algorithm is deterministic, since each module knows its final position at the start. The algorithm is distributed because each module uses local contact information and its current location to determine its ability to move in each round of the execution. There is no other inter-module communication during the execution.

A. Algorithm Assumptions

1. Each module knows the total number of modules in the system, n, and the goal configuration, G.
2. Initially, one module is in each cell of I.
3. I is a straight chain.
4. G is an admissible configuration and there is a particular admissible substrate path that is known to all the modules.
5. I and G overlap in goal cell $G_{1,1}$, as described in Section V-A.

To simplify the presentation of our reconfiguration algorithm, we assume the coordinates of G are ordered at each module as follows:

- The coordinates of cells on the substrate path are stored in a list, in the order in which the cells occur on the directed path from G_1 to G_m.
- The coordinates of cells in G that are north and south of the substrate path are stored in separate lists in the order that these cells will be filled by modules in I (see figure 12 for an example).

B. Distributed Reconfiguration Algorithm

The algorithm works in synchronous rounds. In each round, each module calculates whether it is free and all free modules move simultaneously. In figure 10, the modules labeled *trapped* are unable to move due to hardware constraints and those labeled *free* represent modules that are allowed to move in our algorithm, possibly after some initial delay. The modules in the *other* category are restricted from moving by our algorithm, not by hardware constraints.

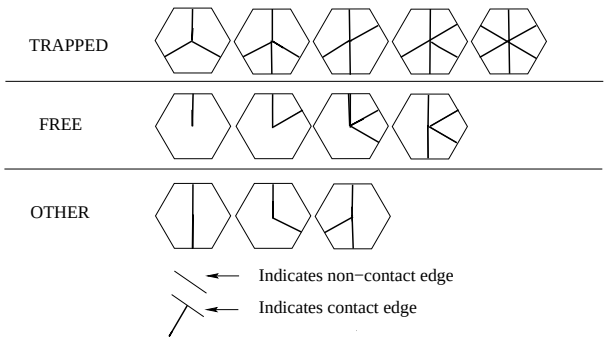

Fig. 10. Contact patterns possible in algorithm.

Modules in I initially calculate their position in I, direction of rotation, possible delay and final coordinates in G by determining their lattice distance from cell $G_{1,1}$. A module calculates the goal cell it will occupy by comparing its position in I to the length of the arrays of coordinates on, north, and south of the substrate path.

Let p be the admissible substrate path, starting with the cell that has an edge incoming from cell $G_{1,1}$. Modules in positions $\leq |p|$ fill in the substrate path first. After p is filled, modules alternate rotation directions, filling the columns projecting north and south of p from east, G_m, to west, G_1. Figure 12 has numbered goal cells showing how initial module positions correspond to final goal positions.

As in our previous paper[23], modules use specific patterns of rotation and delay in our algorithm, as listed below.

1. *(1,0)-bidirectional*: modules alternate direction with delay of 1 time unit after free for modules in positions > 1 rotating clockwise (CW) and no delay after free for modules rotating counterclockwise (CCW).
2. *2-unidirectional*: modules rotate same direction with delay of 2 after free for modules in positions > 1.

Reconfiguration Algorithm

- For modules in positions 1 through $|p|$:
  - Modules use *2-unidirectional* pattern, starting in CW direction.
- For modules in positions $> |p|$:
  - Modules use *(1,0)-bidirectional* pattern until all cells on one side of p are filled. After this, modules use *2-unidirectional* pattern, with either CW or CCW direction, depending on whether there are cells remaining to be filled on the north or south side of p.
- Once a module starts moving, it moves in every round thereafter until it occupies the goal cell it has calculated as its final position. When a module is in the goal cell it should occupy, it stops.
- Once a module stops in the goal cell it should occupy for a round it never moves out of that goal cell.

Figure 12 depicts an example execution of the reconfiguration algorithm.

C. Analysis of Reconfiguration Algorithm

The distributed reconfiguration algorithm described in this section starts with a straight chain to (not necessarily straight) chain reconfiguration, with the first $|p|$ modules filling in the admissible substrate path, p. From the proof sketch in Section V-C, it can be seen that the moving modules (separated by 2 spaces) will not become deadlocked (i.e., by moving into a position where they have a contact pattern that is not "free" (cf. Figure 10)) prior to reaching their calculated goal positions. Since the modules filling in p all move south on the east side of I, it is not possible for them to change order and collide. Once the modules in p are in place, other moving modules are separated by p, ruling out possible collisions.

VII. Simulation Results

Our simulation experiments were inspired by the work of Pamecha et al. [16], where configurations of similar shape but varying number of modules were used to evaluate their algorithm. Direct comparison of the complexity of the algorithms presented in this paper with the results obtained by the centralized reconfiguration algorithm of Pamecha et al. is not possible due to the fact that their simulations involved the reconfiguration of arbitrary shapes of I to arbitrary shapes of G.

We developed an object-oriented discrete event simulator to test the reconfiguration algorithms. Initially, the goal coordinates are specified and each module in I performs the calculations to determine its target goal position, depending on its initial position. During each round,

the simulator checks the local status of every module, and then moves all eligible modules in the same step, thereby accurately simulating a real distributed system.

A. Effect of Heuristics in FIND_PATH

We first experimented with running our algorithm on various shapes using different numbers of modules, testing the effect on performance of varying the heuristic choice in rule 2 of the FIND_PATH algorithm. Performance is measured in terms of number of rounds and number of moves needed for the reconfiguration.

The shapes experimented on included: 1) wedges of similar orientation and variable size, 2) rectangles that lengthened on the E-W axis while remaining fixed on the N-S axis, and 3) diamonds of similar orientation and variable size. These shapes were chosen because they are simple and yet illustrative of how heuristics can affect the performance of the reconfiguration algorithm.

The first heuristic (SN for "select north") chose the NE edge whenever there was a choice of NE or SE edges, biasing the substrate path to "hug" the north side of G. The second heuristic (SS) used a "seesaw" pattern, selecting the edge in the opposite direction as the edge last selected when there was a choice. The third heuristic (GR) used a greedy strategy in which the edge to the NE or SE was selected based on whichever choice most evenly divided the next column to the east.

Figure 11 illustrates the paths found by the SN heuristic, the SS heuristic, and the GR heuristic for a wedge of 29 cells, a rectangle of 21 cells, and a diamond of 26 cells. Heuristic GR was able to more evenly split G into halves for each shape when n was sufficiently large.

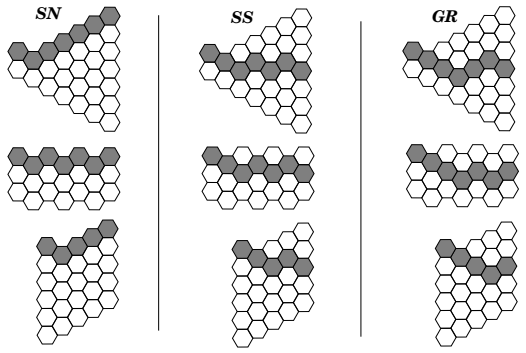

Fig. 11. Example paths found for SN, SS, and GR heuristics.

Figure 12 shows twelve snapshots of the execution of the reconfiguration algorithm on a wedge of eleven cells after the GR heuristic is used to find the substrate path. Time is advancing from snapshot (a) to (l). Figure 12(a) shows the initial positions of the modules in I in relation to the goal positions. Parts (b) through (d) show the *2-unidirectional* pattern with CW rotation used by modules that will fill in the substrate path. In parts (e) through (h), modules use the *(1,0)-bidirectional* pattern of rotation and delay to fill in the goal cells above and below the substrate path. Finally, in parts (i) though (l), modules finish filling in the goal cells below the substrate path using the *2-unidirectional* pattern with CCW rotation.

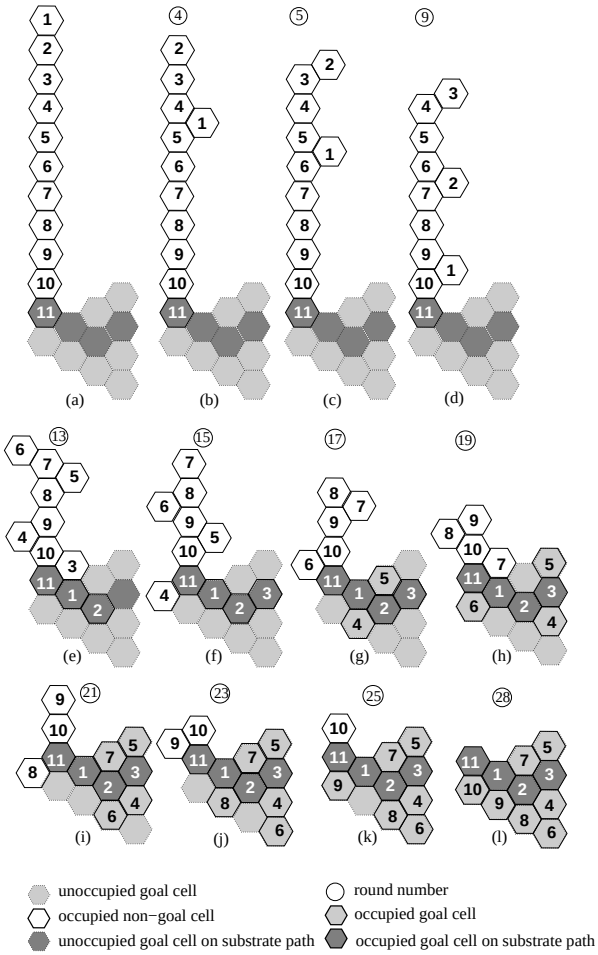

Fig. 12. Example execution of the reconfiguration algorithm.

In Figures 13 and 14(a), we depict the results obtained with wedges of similar orientation and increasing size. Figures 13 and 14(b) show the results of similar experiments on lengthening rectangles and Figures 13 and 14(c) show the results for diamond shapes. These figures show that the number of moves was nearly the same for each heuristic for a given value of n and, for each shape, the number of moves increased more than linearly for increasing values of n. For each of the shapes, when $n > 9$, heuristic GR used fewer rounds than did the SN or SS heuristics. Performance, in terms of number of rounds used, improves when the substrate path evenly divides G because modules can alternate direction, allowing more modules to move in parallel.

Therefore, while any admissible directed path of marked nodes may be chosen as the substrate path, heuristics can improve the number of rounds, and, to a lesser extent, the number of moves, required for reconfiguration.

B. Simulation on Realistic Shapes

Metamorphic robotic systems need to assume different useful shapes. Possible useful shapes include bridges to

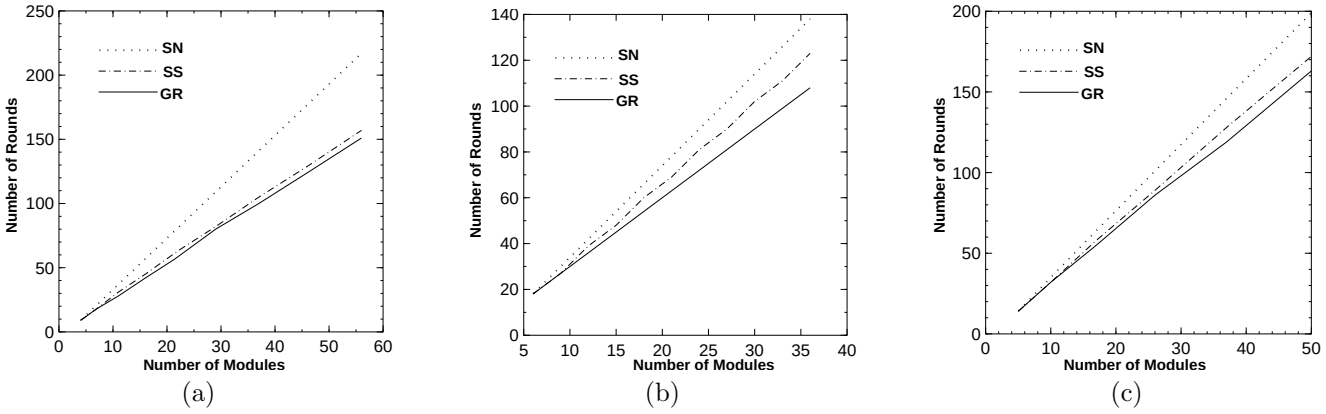

Fig. 13. Rounds used for wedge (a), lengthening rectangle (b), and diamond shaped (c) configurations. Heuristics used are "Select North" (SN), "See-Saw" (SS), and "Greedy" (GR).

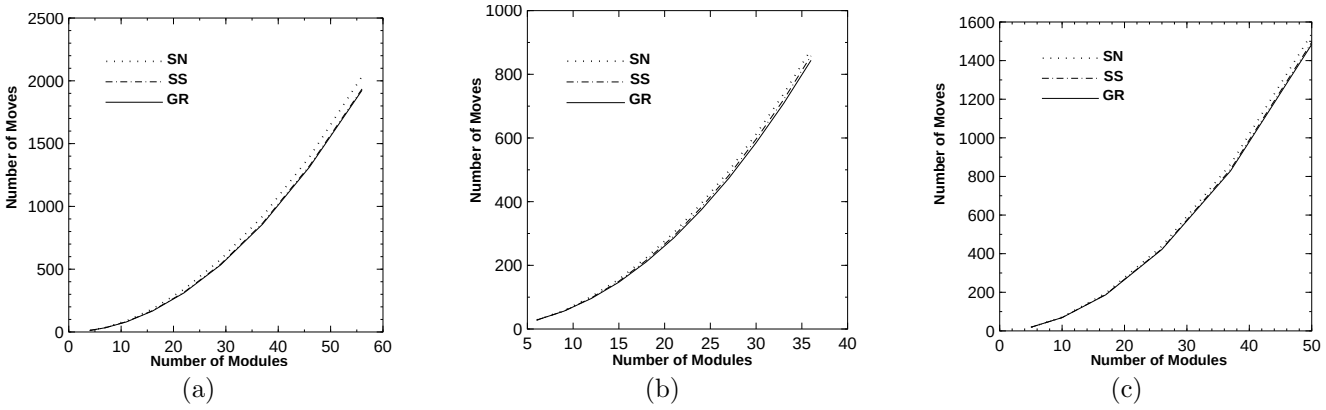

Fig. 14. Moves taken for wedge (a), lengthening rectangle (b), and diamond shaped (c) configurations. Heuristics used are "Select North" (SN), "See-Saw" (SS), and "Greedy" (GR).

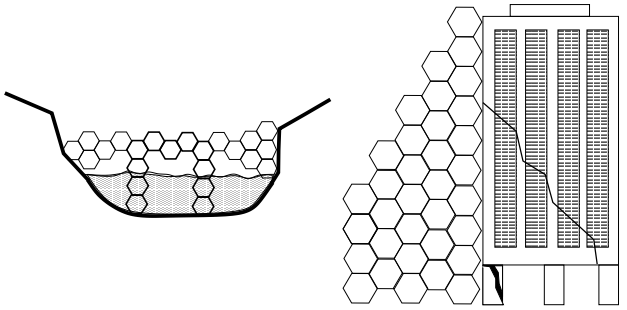

Fig. 15. Metamorphic system as bridge (left) and support for building (right).

span rivers or rough terrain and buttresses to support collapsing buildings or temporary constructions such as emergency flood levees (see Figure 15). In this section, we present the results of simulation experiments involving the reconfiguration of systems composed of varying numbers of modules into useful structures. Since the greedy heuristic had the best performance in terms of number of rounds when tested on simple shapes, we used the greedy heuristic when finding a substrate path in all experiments in this section.

Figure 16(a) shows three examples of the bridge shape used in our experiments, depicting the basic bridge pattern for 11, 18 and 25 modules. The shaded modules represent the substrate path chosen by our algorithm using the greedy heuristic. As shown in Table I, we continued the experiment by increasing the number of modules simulated to over 50. The extension of the bridges for higher number of modules follows the pattern depicted in Figure 16(a).

Figure 16(b) shows two examples of the tall buttress shape used in our experiments, depicting the basic pattern for 19 and 24 modules. The shaded modules represent the substrate path chosen by our algorithm using the greedy heuristic. As shown in Table I, we continued the experiment by increasing the number of modules simulated to about 50. The extension of the buttresses for higher number of modules follows the pattern depicted in Figure 16(b).

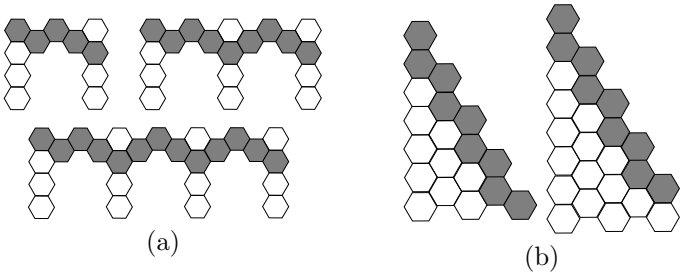

Fig. 16. Example bridge shapes (a) and tall buttress shapes (b).

Table I shows that the number of rounds used in the reconfigurations from chains to more "realistic" shapes in-

TABLE I
NUMBER OF ROUNDS AND MOVES USED FOR BRIDGE AND BUTTRESS.

Shape	Modules	Rounds	Moves
Bridge	11	36	89
	18	61	239
	25	86	466
	32	111	770
	39	136	1151
	46	161	1609
	53	186	2145
Tall Buttress	19	74	269
	24	95	418
	29	116	597
	34	137	806
	44	178	1311

creases in a linear fashion as did the number of rounds for "simple" shapes in the last section. The increase in the number of moves as the number of modules increases is also similar to that shown for the "simple" shapes. Clearly, filling in each of these "realistic" shapes from the bottom up would be preferable if gravity were a concern in the reconfiguration. The substrate paths chosen do not attempt to follow this bottom-up pattern. The shapes considered in this section also do not lend themselves to effective use of the heuristics in FIND_PATH, resulting in low overall parallelism in these simulations. However, these shapes do represent "real world" applications for metamorphic robots and we have demonstrated that our algorithm can effectively perform the reconfigurations.

In [23], we showed that our distributed algorithm for straight chain to chain reconfiguration in a system of hexagonal robots, given our system assumptions, takes $O(n)$ rounds and $O(n^2)$ moves. The experimental results we present in this section suggest that our straight chain to admissible goal reconfiguration algorithms have similar complexity.

VIII. Conclusions and Future Work

The algorithms presented in this paper rely on total knowledge of the goal configuration. Each module precomputes all aspects of its movement once it has sufficient local information to reconstruct the entire initial configuration. We proved the correctness of our centralized algorithm for finding a substrate path and tested the performance of our distributed reconfiguration algorithm through simulation.

The orientation of the initial chain to the admissible goal shape limits the possible choices for the point of contact between the initial and goal configurations and may not be an efficient planning technique in many situations. In our recent work [21], we present algorithms that do not place such a strict orientation criteria on the initial positions of the chain and the admissible goal configuration. The algorithms presented in [21] use heuristics to improve the time used for reconfiguration by choosing substrate paths that allow faster reconfiguration. For example, if the substrate path is a straight chain to the SE or NE, it can be filled using a pattern in which modules alternate direction, as was done in our straight chain to straight chain algorithms[23]. Although we do not consider the presence of obstacles in this paper, we do consider reconfiguration in the presence of obstacles in a more recent paper [21]. We are currently working on improvements to these "obstacle-robust" algorithms.

Since we restrict the initial configuration to a straight chain, it is rather simple for the modules to reconstruct the entire initial configuration. We believe that a more flexible approach will be helpful in designing reconfiguration algorithms for more irregular configurations, more asynchronous systems, and those with unknown obstacles. Part of such a flexible approach will include the ability for modules to detect and resolve collisions and deadlock situations when they occur, rather than precomputing trajectories that avoid these situations. We have some initial ideas for ways to deal with module collision and deadlock on the fly, which we leave for future work.

Our algorithms assume that the modules are capable of a combined rigid body rotation coupled with changing joint angles, enabling them to crawl into and out of tight spaces (e.g., between two stationary modules). If the modules are rigid and have to roll around other modules, they would not be capable of moving into and out of such small spaces. The modification of our algorithms for alternate hardware specifications is left for future work.

Other open problems that we have not addressed in this paper include the generalization of our algorithms to three dimensional modules and planar modules in three dimensions, inclusion of force and load-bearing capabilities into module assumptions, and consideration of other hardware constraints on module movements.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for helpful comments and suggestions that contributed to the presentation of this paper.

References

[1] A. Casal and M. Yim. Self-reconfiguration planning for a class of modular robots. In *Proc. of SPIE Symposium on Intelligent Systems and Advanced Manufacturing*, vol. 3839, pages 246–256, 1999.

[2] C.-J. Chiang and G. Chirikjian. Similarity metrics with applications to modular robot motion planning. *Autonomous Robots Journal, special issue on self-reconfiguring robots*, Vol. 10, No. 1, pages 91–106, Jan. 2001.

[3] G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 449–455, 1994.

[4] G. Chirikjian. Metamorphic hyper-redundant manipulators. In *Proc. of Intl. Conf. on Advanced Mechatronics*, pages 467–472, Aug. 1993.

[5] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of Robotic Systems*, Vol. 13, No. 5, pages 317–338, May 1996.

[6] T. Fukuda, M. Buss, H. Hosokai, and Y. Kawauchi. Cell structured robotic system CEBOT (control, planning and communication methods). *Intelligent Autonomous Systems*, Vol. 2, pages 661-671, 1989.

[7] G. J. Hamlin and A. C. Sanderson. *Tetrobot: A modular approach to reconfigurable parallel robotics*. Kluwer Academic Publishers, Newton, MA, 1997.

[8] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with

morphogenesis in a vertical plane. In *IEEE Intl. Conf. on Robotics and Automation*, pages 2858–2863, May 1998.

[9] K. Kotay and D. Rus. Motion synthesis for the self-reconfiguring molecule. In *IEEE Intl. Conf. on Robotics and Automation*, pages 843–851, 1998.

[10] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule: design and control algorithms. In *Workshop on Algorithmic Foundations of Robotics*, pages 376–386, 1998.

[11] W. H. Lee and A. C. Sanderson. Dynamic analysis and distributed control of the tetrobot modular reconfigurable robot system. *Autonomous Robots Journal, special issue on self-reconfiguring robots*, Vol. 10, No. 1, pages 67–82, Jan. 2001.

[12] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 441–448, 1994.

[13] S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji. Self-assembling method for mechanical structure. In *Artif. Life Robotics*, Vol. 1, pages 111-115, 1997.

[14] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 432–439, 1998.

[15] A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.

[16] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 4, pages 531–545, Aug. 1997.

[17] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots Journal, special issue on self-reconfigurable robots*, Vol. 10, No. 1, pages 107–124, Jan. 2001.

[18] B. Salemi, W.-M. Shen, and P. Will. Hormone-controlled metamorphic robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 4194–4199, May 2001.

[19] C. Unsal, H. Kiliccote, M. E. Patton, and P. Khosla. Motion planning for a modular self-reconfiguring robotic system. *Distributed Autonomous Robotic Systems 4*, Springer, Nov. 2000.

[20] J. Walter. Distributed algorithms for mobile computing systems. Doctoral dissertation, Dept. of Computer Science, Texas A&M University, Dec. 2000.

[21] J. Walter, E. Tsai, and N. Amato. Choosing good paths for distributed reconfiguration of hexagonal metamorphic robots. To appear in *Proc. of IEEE Intl. Conf. on Robotics and Automation*, May 2002.

[22] J. Walter, J. Welch, and N. Amato. Distributed reconfiguration of hexagonal metamorphic robots in two dimensions. In *Proc. of SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems*, Vol. 4196, pages 441–453, 2000.

[23] J. Walter, J. Welch, and N. Amato. Distributed reconfiguration of metamorphic robot chains. In *Proc. of ACM Symp. on Principles of Distributed Computing*, pages 171–180, 2000.

[24] M. Yim. A reconfigurable modular robot with many modes of locomotion. In *Proc. of Intl. Conf. on Advanced Mechatronics*, pages 283–288, Aug. 1993.

[25] M. Yim, J. Lamping, E. Mao, and J. G. Chase. Rhombic dodecahedron shape for self-assembling robots. SPL TechReport P9710777, Xerox PARC, 1997.

[26] Y. Zhang, M. Yim, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots Journal, special issue on self-reconfigurable robots*, Vol. 10, No. 1, pages 41–56, Jan. 2001.