Enveloping Obstacles with Hexagonal Metamorphic Robots

Jennifer E. Walter
Vassar College
walter@cs.vassar.edu

Elizabeth M. Tsai
Swarthmore College
tsai@cs.swarthmore.edu

Nancy M. Amato
Texas A&M University
amato@cs.tamu.edu

***Abstract*— The problem addressed is the distributed reconfiguration of a metamorphic robot system composed of any number of two dimensional robots (modules). The initial configuration we consider is a straight chain of modules, while the goal configuration satisfies a simple admissibility condition. Our reconfiguration strategy depends on finding a contiguous path of cells, called a *substrate path*, that spans the goal configuration. Modules fill in this substrate path and then move along the path to fill in the remainder of the goal without collision or deadlock.**

In this paper, we address the problem of reconfiguration when a single obstacle is embedded in the goal environment. We introduce a classification for traversable surfaces which allows for coherence in defining admissibility characteristics for various objects in the hexagonal grid. We present algorithms to 1) determine if an obstacle enveloped in the goal fulfills a simple admissibility requirement, 2) include an admissible obstacle in a substrate path, and 3) accomplish distributed reconfiguration.

***Index Terms*—Metamorphic robots, distributed reconfiguration, obstacle**

I. Introduction

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 [5], a subset of self-reconfigurable systems, are further limited by requiring each module to be identical in structure, motion constraints, and computing capabilities and to have a regular symmetry so that they can densely pack the plane to form two and three dimensional solids. In these systems, robots achieve locomotion by moving over a substrate composed of one or more other robots. The mechanics of locomotion depend on the hardware and can include module deformation [6], [12], [13] or rigid motion [1], [7], [10], [11] [19], [20].

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as structural support or tumor excision [12], as well as being useful in environments not amenable to direct human observation and control (e.g. interplanetary space). The complete interchangeability of the robots provides a high degree of system fault tolerance.

The motion planning problem for a self-reconfigurable 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 existing motion planning strategies rely on centralized algorithms to plan and supervise the motion of the system components [5], [8], [9], [12], [13], [18]. Others, such as [2], [10], [11], [19], and [20], use distributed approaches which rely on heuristic approximations or require communication between robots during the reconfiguration process.

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian [6]. We consider a distributed motion planning strategy, given the assumption of initial global knowledge of G. Our distributed approach offers the benefits of localized decision making, the potential for fault tolerance, and 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 [17] or to a goal configuration that satisfies a general "admissibility" condition [15]. In [14], we presented algorithms and heuristics to choose paths for fast reconfiguration.

In this paper we present a new definition of an admissible traversal surface in a hexagonal grid and show that this definition can be readily extended to parallel reconfiguration in the presence of a single obstacle. We present algorithms to first determine whether an obstacle contained in G has a particular clearance that makes it admissible, then to incorporate admissible obstacles in the substrate path, and lastly, to reconfigure the system in a distributed fashion.

A. Related work

In this section, we briefly survey related algorithmic work on metamorphic robots. We group this work according to the general shape of the modules and then discuss the work specifically related to obstacles in the reconfiguration environment.

Hexagonal robots: Chirikjian [6] and Pamecha [12] present centralized reconfiguration algorithms for planar hexagonal modules that use the distance between all modules in I and the coordinates of each goal position to ac-

Amato and Walter supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874, by the Texas Higher Education Coordinating Board grant ARP-036327-017, and by the DOE ASCI ASAP program grant B347886. Tsai supported in part by the CRA-W Distributed Mentorship Program.

complish the reconfiguration of the system. Pamecha et al. [12] define the distance between configurations as a metric and apply this metric to system self-reconfiguration using a simulated annealing technique to drive the process towards completion. Murata et al. [10] use a probabilistic distributed approach to reconfigure a system of hexagonal modules when position of the goal is irrelevant and only final shape matters.

Square robots: Chiang and Chirikjian [4] present bisecting techniques used to find intermediate configurations between any given initial and final configurations. They combine these techniques with the simulated annealing algorithm of Pamecha et al. [12] to reconfigure the system.

Cubic robots: A centralized motion planning strategy for three dimensional cubic robots is presented by Rus and Vona [13]. A set of distributed motion planning algorithms for a system of cubic robots is presented by Butler et al. in [2]. These algorithms allow modules to move asynchronously after a planning phase that requires extensive inter-module communication. Murata et al. [11] present a probabilistic distributed approach to reconfigure a system of cubic modules.

Rhombic dodecahedral robots: Yim et al. [19], Zhang et al. [20] and Bojinov et al. [1] present distributed algorithms to reconfigure rhombic dodecahedral modules. These algorithms are probabilistic and require substantial message passing between neighboring modules.

Obstacle-related reconfiguration: The presence of obstacles in the reconfiguration environment is briefly considered by Chirikjian in [5]. In this approach, a heuristic is used to attract modules to an obstacle so that they converge around it. Bojinov et al. [1] provide a distributed strategy for grasping objects in the environment using rhombic dodecahedral modules by probabilistically "growing" extensions to envelop the obstacle. Butler et al. [3] present a rule set for distributed locomotion of layers of cubic modules over obstacles on the traversal surface.

B. Our approach and problem definition

Our objective is to design a distributed algorithm that will cause the modules to move from an initial configuration, I, in the plane to a known goal configuration, G. This algorithm should ensure that modules do not collide with each other, and the reconfiguration should be accomplished in a minimal number of rounds.

The major differences between our approach and those of other researchers are summarized below:

1) *Our algorithms require no message passing.*
2) *Our algorithms are deterministic, ensuring that the reconfiguration can be accomplished without deadlock or collision provided I is a straight chain and G fulfills simple admissibility requirements.*
3) *Our algorithms are particular to the motion constraints of planar, hexagonal modules.*

In Section II we describe the system assumptions and the problem definition. Section III describes our algorithm for determining admissibility of a traversal surface. Section IV introduces simple admissibility conditions for obstacles and presents a method for reconfiguration in the presence of obstacles. Section VI provides a discussion of our results and future work.

II. System model

The plane is partitioned into equal-sized hexagonal cells and labeled using the same coordinate system as described by Chirikjian [5].

A. 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.

Modules move according to the following rules.

1) Modules move in lockstep rounds.
2) In a round, a module M is capable of moving to an adjacent cell, C_1, iff (see Fig. 1 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, push, or pull 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.

III. Admissible traversal surfaces

In this section we present a classification of traversable surfaces in a hexagonal system under the motion constraints presented in Section II.

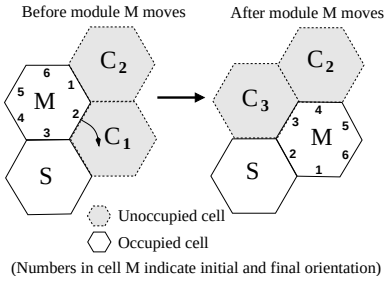

Fig. 1. Before and after module movement. S is substrate module.

In the following definition, let set T represent a particular set of cells in the plane.

Definition 1: The *d-clearance* of a cell $i \in T$ is the number of consecutive cells $\notin T$ that are aligned along the outgoing vector originating in the center of cell i and normal to side d, $d \in \{\text{N, S, NW, SW, NE, SE}\}$.

Figure 2 shows a cell with a *SE-clearance* ≥ 3.

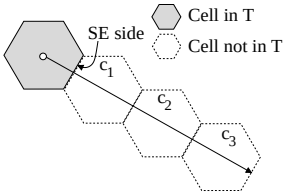

Fig. 2. Cell with *SE-clearance* ≥ 3.

Let s be a contiguous sequence of distinct cells, $c_1, c_2, \ldots, c_k$ from a set T such that each cell is adjacent to the previous. The *adjacency direction* between two contiguous cells c_i and c_{i+1} is the side of c_i that is adjacent to c_{i+1}.

Definition 2: A *segment* of s is a contiguous subsequence of s of length ≥ 2. In a *d-segment*, each cell is direction d of the previous.

Definition 3: A cell $c_i \in s$ is the *end* (resp. *beginning*) of a *d-segment* if cell c_{i+1} (resp. c_i) is adjacent to c_i (resp. c_{i-1}) in some direction not equal to d.

The admissibility conditions for a traversal surface are directly related to the degree of parallelism possible, i.e., how closely modules moving across the surface 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 [17]. However, acute angle intersections are very commonplace in configurations of hexagonal robots. Our definition of an admissible traversal surface is therefore based on configuration surfaces over which moving modules with two empty cells between them can move without collision or deadlock.

We present our definition of an admissible traversal surface for the situation where modules traverse the surface from west to east, and then make a similar definition for the situation where traversal is from east to west.

Definition 4: An *east-monotone admissible traversal surface* is a contiguous sequence $s = c_1, c_2, \ldots, c_k$ of distinct cells from a set T such that

1) no c_i is the end of a NW or SW segment (i.e., each cell is adjacent to the previous, but not to its west),
2) the north side of s is such that
   a) every c_i in a N segment with NW-clearance > 0 has NW-clearance ≥ 3, and
   b) every c_i in a S segment with NE-clearance > 0 has NE-clearance ≥ 3; and
3) the south side of s is such that
   a) every c_i in a S segment with SW-clearance > 0 has SW-clearance ≥ 3, and
   b) every c_i in a N segment with SE-clearance > 0 has SE-clearance ≥ 3.

Figure 3 shows examples of surfaces (depicted by shaded cells) that are composed of a sequence of distinct, contiguous cells, with cells on the north side shaded. The cells in each of these surfaces can be numbered consecutively, from west to east, so that they adhere to Definition 4, part 1. Figure 3, parts (a) and (d) show the north and south sides, respectively, of an east-monotone admissible traversal surface. Parts (b) and (c) show north sides of surfaces that violate parts 2a and 2b of Definition 4, respectively. Parts (e) and (f) show south sides of surfaces that violate Definition 4 parts 3a and 3b, respectively.

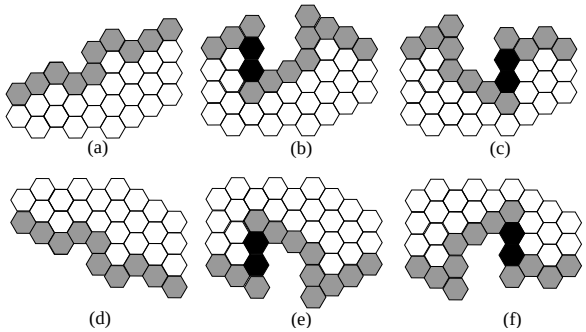

Fig. 3. East-monotone admissible surface (a). In (b), (c), (e), and (f), the black cells violate Definition 4, parts 2a, 2b, 3a and 3b, respectively.

A *west-monotone admissible traversal surface* is defined exactly as the east-monotone admissible surface except that in Definition 4, part 2, NW is substituted for NE and vice versa, and SE is substituted for SW and vice versa in Definition 4, part 3.

A. Admissible goal configurations

In this section we define admissible goal configurations and describe a centralized algorithm that tests whether a given configuration is admissible, i.e., whether it contains

an *admissible substrate path*. Note that we do not consider goal configurations that contain obstacles in this section, as the admissibility of the goal is determined independent of obstacle admissibility. The presence of obstacles in the goal environment, as well as conditions for obstacle admissibility, are discussed in Section IV.

Without loss of generality, assume I is a straight chain that intersects G in exactly one cell on the perimeter of G. The number of modules in I and the number of cells in G is n.

Let $G_1, G_2, \ldots, G_m$ be the columns of G, such that G_1 is the column in which I intersects G and G_m is the column furthest from column G_1. Suppose that G is oriented such that column G_1 is the westmost column, G_m is the eastmost column, and each column of G is a contiguous straight chain oriented north-south. Figure 4 shows how the columns of G are labeled.

The assumptions concerning the relative positions of I and G can be made without loss of generality because if I is a straight chain that is not intersecting G, then the algorithms presented in [17] for straight chain to straight chain reconfiguration can be used to reorient I in relation to G. I is assumed to be a straight chain only because this configuration is well-defined. We intend to relax this assumption in future work.

Definition 5: Let a *path* p be a contiguous sequence of distinct cells, $c_1, c_2, \ldots, c_k$.

In the remainder of this paper, north and south segments of p may be referred to as *vertical* segments when it is not necessary to be more specific about direction.

Definition 6: p is an *admissible substrate path* if

1. p begins with the cell in which I and G overlap,
2. subsequent cells are all in G,
3. p spans G, from column G_1 to column G_m,
4. each c_i in p forms the beginning of a vertical segment only if there is no goal cell to the NE or SE of c_{i-1}, and
5. both the north and south sides of p form an *east-monotone admissible traversal surface* with respect to set T, the set of all cells in G.

Definition 7: G is an *admissible goal configuration* if there exists an admissible substrate path in G.

Figure 4 depicts an example of an admissible (a) and an inadmissible (b) configuration of G.

B. Finding substrate paths

Our procedure for finding an admissible substrate path in G was presented in [14] and [15]. In this paper, we only briefly review this procedure.

- Label the columns of G as described in the beginning of this section, with the cells in each G_i labeled $G_{i,1}$, $G_{i,2}$,..., from north to south.

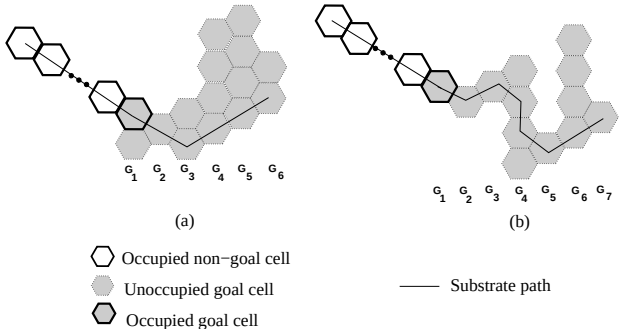

Fig. 4. Example admissible (a) and inadmissible (b) G.

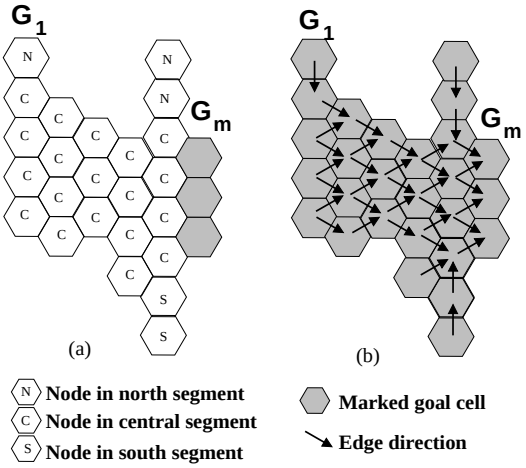

Fig. 5. Directed graph H formed by algorithm.

- Represent each goal cell as a node in a graph H. Initially there is an undirected edge between each pair of adjacent goal cells.

Once H has been initialized, the edges are directed by marking the vertices that are determined to have an admissible path to a goal cell in the eastmost column. This is accomplished by the DIRECT_EDGES algorithm as follows:

- First, every node in column G_m is marked, as shown in Figure 5(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 5(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 (resp. SE) edge would be directed toward a neighbor with an outgoing S (resp. N) edge (eg., Figure 5(b), where goal cell $G_{4,4}$ is marked, but its SE edge is not*

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

Once H has been constructed (as shown in Figure 5(b)), a graph traversal algorithm is used to find all admissible substrate paths from column G_1 to column G_m in the graph. A weighting heuristic that favors straight and single-bend paths (i.e., those that allow for maximum parallel movement of modules) is used in conjuction with the graph traversal to rank the candidate paths. The substrate path used for reconfiguration is then selected from this ranked set by using a second heuristic that gives preference to paths that most evenly bisect the goal configuration. During reconfiguration, modules fill the substrate path first, then move along both sides of the path to fill in the rest of G in columns from E to W and from the path outward to the N and S.

IV. Obstacles

In this section, we consider the presence of a single obstacle in the coordinate system and present a strategy for reconfiguration when obstacles are present in the goal.

An obstacle is a sequence of one or more "forbidden cells" that modules cannot enter. We consider obstacles that are composed of hexagons of the same size as the cells of the plane and we assume each hexagon in the obstacle occupies exactly one of the cells. Since at this point, we consider only the presence of obstacles that are contained completely within G, the cells surrounding the obstacle are goal cells. Modules may touch obstacle perimeter cells and may use them as substrate for movement.

Definition 8: An obstacle is *admissible* if

1) it is completely enclosed by the cells of G, and
2) every obstacle cell has infinite *d-clearance* for each side d with *d-clearance* > 0, where the set T is defined as the set of all obstacle cells.

Definition 8 requires obstacle cells on the perimeter of an admissible obstacle to have an infinite *d-clearance* with respect to other obstacle cells. Here, the set T from Definition 1 includes only the obstacle cells because we want to ensure that the obstacle surface contains no holes or pockets where modules may become trapped. We are currently working on algorithms to determine the proper order in which to fill cells in holes on an obstacle surface but that is a subject for future work.

A. Reconfiguration with admissible obstacles inside G

Even if an obstacle contained in G is admissible, our reconfiguration algorithms must ensure that narrow "pockets" do not form on the the obstacle perimeter during reconfiguration.

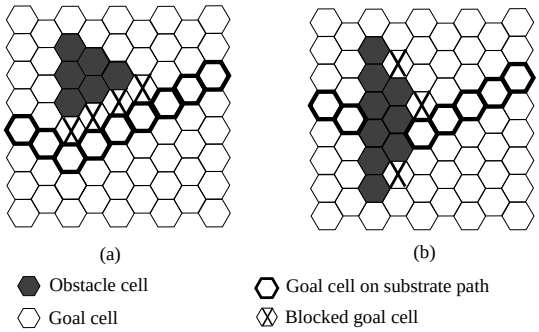

Fig. 6. Filling in G using substrate path that does not pass through obstacle. Part (a) shows an admissible obstacle, a possible substrate path in G and the goal cells that could not be filled in due to lack of clearance for module movement. Part (b) shows another admissible obstacle, a possible substrate path found through G, and goal cells that would be blocked during reconfiguration.

Situations where pockets can form during reconfiguration are shown in Figure 6. In part (a), some cells between the substrate path and the obstacle are blocked after the substrate path is filled. This problem is easily solved by requiring the substrate path to pass through the obstacle, thereby forming a combination obstacle and substrate path traversal surface. Figure 6(b) shows a possible substrate path chosen by the algorithms presented in Section III. This path passes through the obstacle, and forms a traversable path to the west of the obstacle. Because the substrate path is filled first and then the remaining goal cells are filled in columns from east to west and from the substrate path outward, the goal cells marked with "X"s in Figure 6(b) will not be filled due to motion constraints on the modules.

To avoid this problem, we want the obstacle to "taper" from west to east, so that the eastmost column contains a single cell. Figure 7(a) shows an obstacle that has been augmented with "repaired" goal cells which will be filled in, from west to east, before the rest of the goal to the east of the obstacle is filled. Part (b) of Figure 7 shows a possible admissible substrate path adjoined to the repaired obstacle. The algorithm to repair obstacles prior to reconfiguration is presented in the following section.

B. Algorithm to repair obstacles

Obstacles are repaired in a single pass, from west to east, by identifying goal cells in the "cone" to the east of the obstacle and adding them to a set of "repaired" cells.

Let G be oriented as described in Section III and let t be the number of columns in the obstacle. Let $R_1, R_2, \ldots, R_t$ be the columns of the obstacle, labeled from west to east.

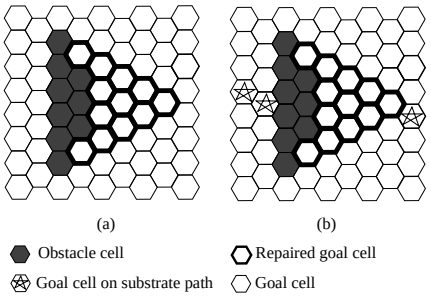

Fig. 7. Repaired obstacle in G (a). Part (b) shows the repaired obstacle and a possible substrate path found in this configuration.

Initially, the only repaired cells are the obstacle cells themselves. For each column R_i ($1 \leq i \leq t$), number the obstacle cells from north to south, starting with 1 for the northmost cell in each column. The algorithm converts the surface of the obstacle to an east-monotone admissible traversal surface by adding repaired cells if possible. The algorithm checks existing repaired cells with NE- or SE-clearance > 0 for NE- or SE-clearance ≥ 3 with respect to T, the set of all goal and obstacle cells. The set T includes both goal and obstacle cells in this case because we must consider the clearance that modules moving over the repaired obstacle will have when goal cells to the east are filled. Obstacle repair is accomplished as follows:

For each column R_1 to the eastmost column containing repaired cells, examine each cell, $R_{i,j}$, in the column from north to south.

- If $R_{i,j}$ has an obstacle or repaired cell to the N: if there is a goal cell to the NE, mark the NE cell as repaired; else if there is not a goal cell to the NE and $R_{i,j}$ does not have NE-clearance ≥ 3, the obstacle is unrepairable.
- If $R_{i,j}$ has an obstacle or repaired cell to the S: if there is a goal cell to the SE, mark the SE cell as repaired; else if there is not a goal cell to the SE and $R_{i,j}$ does not have SE-clearance ≥ 3, the obstacle is unrepairable.

Continue to repair the obstacle until the "cone" shape is achieved (i.e., until all obstacle columns have been processed and either $R_{i,1}$ has no more repaired cells to the N or S or all cells in the eastmost column of R have NE- and SE-clearance ≥ 3) or until the obstacle is found to be unrepairable.

Figure 8 shows snapshots taken during the execution of the repair of an obstacle, with time progressing from (a) to (c).

Figure 9(a) shows a goal configuration containing an admissible obstacle that is not repairable. In Figure 9(b), cell $R_{3,1}$ has a repaired cell to the S, but the cell to the SE of cell $R_{3,1}$ is not a goal cell and $R_{3,1}$ does not have SE-clearance ≥ 3 with respect to the set of all goal cells (T).

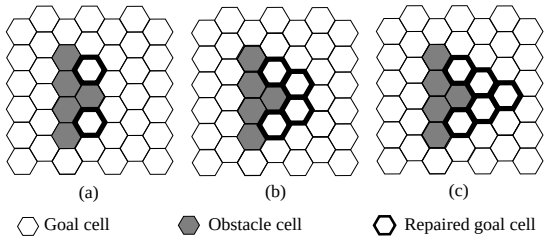

Fig. 8. Example obstacle repair. In part (a), cells $R_{2,1}$ and $R_{2,3}$ are repaired when $R_{1,1}$ and $R_{1,4}$ are processed. In part (b), cells $R_{3,1}$ and $R_{3,2}$ are repaired when $R_{2,1}$ and $R_{2,3}$ are processed. In part (c), cell $R_{4,1}$ is repaired when cell $R_{3,1}$ is processed.

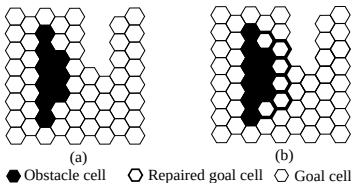

Fig. 9. Obstacle in G (a) and unrepairable obstacle (b).

V. Distributed reconfiguration algorithm

In this section, we describe the distributed algorithm that performs the reconfiguration of I to G after the goal and the obstacle are determined to be admissible and the obstacle surface is repaired.

The overall reconfiguration proceeds as outlined in Figure 10.

if obstacle contained in G is admissible and repairable
1. find *asp* from G_1 to westmost column of obstacle.
2. find *asp* from eastmost cell of repaired obstacle to G_m.
3. fill in cells of G in the following order:
   a) *asp* west of obstacle, west to east,
   b) repaired cells north and south of obstacle, west to east,
   c) repaired cells east of obstacle, west to east,
   d) *asp* east of obstacle (if it exists), and
   e) north and south of *asp*/obstacle, from east to west.

else report failure.

Fig. 10. Schema for reconfiguration with obstacles (*asp* is an abbreviation for admissible substrate path).

In step 1 of the reconfiguration schema in Figure 10, algorithm DIRECT_EDGES is run, beginning by marking the cells in the westmost column of the obstacle instead of the cells in G_m. This ensures that the substrate path will abut against the westmost column of the obstacle. The western substrate path is then selected using the algorithms described in Section III. Step 2 uses algorithm DIRECT_EDGES to direct the edges from the cell in the eastmost column of the repaired obstacle to column G_m. This step is unnecessary if the repaired cells extend into column G_m.

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. G is an admissible goal configuration (note that obstacle cells are treated as goal cells in check for goal admissibility.)
4. The obstacle contained in G is a repaired admissible obstacle.
5. I and G overlap in one goal cell in column G_1, as described in Sect. III.

B. Overview of algorithm

The algorithm works in synchronous rounds. In each round, each module determines whether it is free (cf. Fig. 11). In this figure, 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.

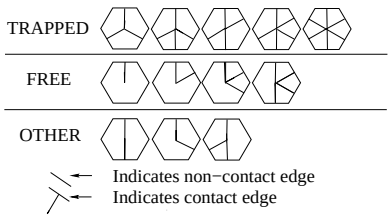

Fig. 11. Contact patterns possible in algorithm.

All modules except module 1 dynamically calculate their position in I, direction of rotation, possible delay and final coordinates in G by counting the modules in initial positions further from the intersection of I and G as they pass and noting the direction (CW or CCW) in which the passing modules rotate.

Let f be the array of coordinates of goal cells (stored locally at each module) starting with the cell that has an edge incoming from the cell in which I and G intersect in column G_1. The cells are inserted in f in the following order:

1. cells in the substrate path west of the obstacle, from west to east,
2. repaired cells north of the obstacle, in columns from west to east and ordered within the columns from south to north,
3. repaired cells south of the obstacle, in columns from west to east and ordered within the columns from north to south,
4. repaired cells east of the obstacle, in columns from west to east and ordered within the columns from south to north, and
5. cells on the substrate path east of the obstacle (if they exist), from west to east.

Coordinates of goal cells to the north and south of the substrate path are also stored in arrays at each module. A module calculates the goal cell it will occupy using its position in I, the length of the arrays of coordinates on, north, and south of the cells in f, and the current count of modules that have passed on both sides.

Modules fill in f first, with module 1 filling the first cell in ordered array f, module 2 filling the second cell, and so on. After every goal cell in f is filled, modules alternate rotation directions, filling the columns projecting north and south of f from east, G_m, to west, G_1.

Figure 12 shows snapshots of the execution of the distributed algorithm. Initially, only module 24 in the straight chain I intersects G and I is slanted from NW to SE. Figure 12(a) is a snapshot of the execution taken after round 4, when modules 1 and 2 have started moving. Parts (b) through (f) are snapshots after rounds 25, 33, 38, 42, and 65, respectively.

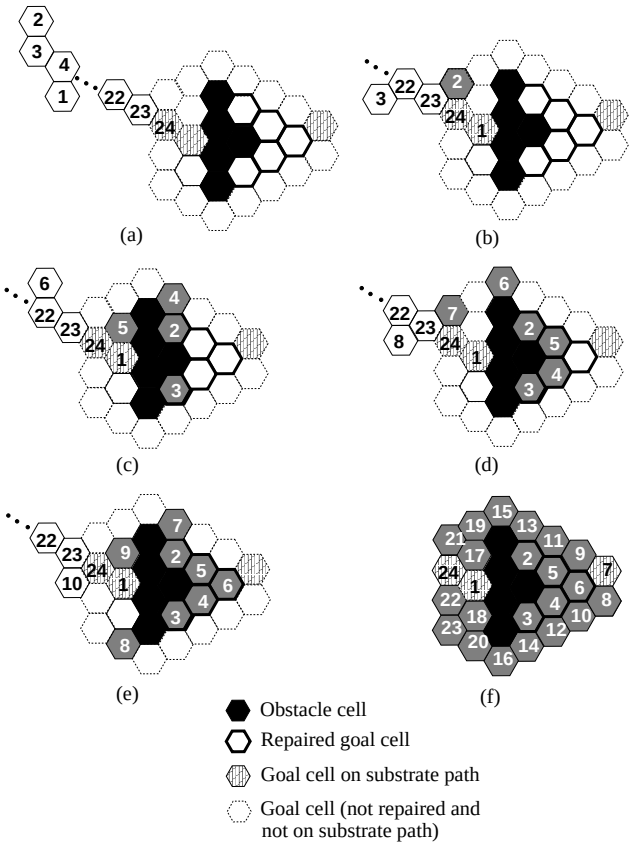

Fig. 12. Snapshots of execution of distributed algorithm.

Modules use the rotation patterns listed below.

1. *unidirectional*: Module i rotates in the same direction as module $i-1$.
2. *bidirectional*: Module i rotates in the opposite direction as module $i-1$.

Each module maintains a variable *delay*, the number of time units a module waits after it is free (cf. Figure 11) and before it makes its first move. Initially, *delay* = 0. The choice of module rotation and and value of *delay* depends on its position in I:

• For modules 1 through $|f|$ (i.e., those modules filling in the substrate path and repaired cells):

– Module 1 has *delay* = 0; modules $2 \ldots |f|$ have *unidirectional* rotation and *delay* = 2 unless otherwise noted.

– If module position corresponds to a position in f on the west substrate path,
  - if the west substrate path meets the obstacle at a NW to SE slant, rotation = CCW.
  - else if the west substrate path meets the obstacle at a SW to NE slant, rotation = CW.

– For the lowest module position i corresponding to ...

... a cell in f north of obstacle, start CW. If module $i-1$ went CCW, *delay* = 1.

... a cell in f south of obstacle, start CCW. If module $i-1$ went CW, *delay* = 1.

... a cell in f east of obstacle, start CW. If module $i-1$ went CCW, *delay* = 1.

... a cell in f on the east substrate path, start CW. If module $i-1$ went CCW, *delay* = 1.

• For modules in positions $|f|+1 \ldots n-1$: modules use *bidirectional* rotation with *delay* = 1 for modules rotating CW and *delay* = 0 for those rotating CCW until all cells either north or south of f are filled. After this, modules use *unidirectional* pattern, with either CW or CCW direction and *delay* = 2.

• The module in position n does not move.

• Once a module stops for a round in the goal cell it has calculated it should occupy, it never moves out of that goal cell.

The pseudocode for the distibuted reconfiguration is the same as that used when no obstacles occur in the goal environment. Since this pseudocode was presented in detail in [14], we only briefly review it here.

In each round, every module that is not already in its calculated final goal cell checks its contact pattern (cf. Figure 11), i.e., on which sides it contacts other modules. In the first round a module detects that it has a free contact pattern, it calculates the value of *delay* according to its position, as described above. If *delay* = 0, the module moves, also choosing its rotation direction according to its position as described earlier; otherwise, if *delay* > 0, the module decrements *delay* and does not move in that round. After a module begins moving, it moves in every round until it occupies the goal cell that it has calculated as its final position.

VI. Conclusions and future work

We have presented a classification of admissible traversal surfaces in the hexagonal grid which allows for generality and coherence when defining admissibility conditions for various objects in the environment. We also addressed the problem of reconfiguration in the presence of a single obstacle enveloped by the goal, using the definition of admissible traversal surfaces to repair the surface of the obstacle prior to distributed reconfiguration.

In this paper, the obstacle admissibility requirements were very simple. In our current work, we are extending the definition of admissible surfaces to include embedded obstacles with more complex surfaces. We believe that the ability to repair an obstacle surface will also be helpful in designing reconfiguration algorithms for goal configurations containing multiple obstacles.

References

[1] H. Bojinov, A. Casal, and T. Hoag. Emergent structures in modulular self-reconfigurable robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, Vol. 2, pages 1734–1741, 2000.

[2] Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. In *Proc. of IEEE Intl. Conf. on Intelligent Robots and Systems*, pages 790–796, 2001.

[3] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 809–816, May 2002.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.

[13] 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.

[14] J. Walter, B. Tsai, and N. Amato. Choosing good paths for fast distributed reconfiguration of hexagonal metamorphic robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 102-109, May 2002.

[15] J. Walter, J. Welch, and N. Amato. Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. Accepted to *IEEE Transactions on Robotics and Automation*, 2002.

[16] J. Walter, B. Tsai, and N. Amato. Concurrent reconfiguration of hexagonal metamorphic robots: Algorithms for fast execution and obstacle envelopment. Submitted, 2002.

[17] 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.

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

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

[20] Y. Zhang, M. Yim, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots Journal, Special Issue on Self-Reconfigurable Robots*, 2000.