Distributed reconfiguration of hexagonal metamorphic robots in two dimensions

Jennifer E. Walter[a], Jennifer L. Welch[a], and Nancy M. Amato[a]

[a] Department of Computer Science, Texas A&M University, College Station, TX 77843-3112.

ABSTRACT

The problem addressed is the distributed reconfiguration of a metamorphic robotic system composed of any number of two dimensional hexagonal modules from specific initial to specific goal configurations. The initial configuration considered is a straight chain of 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. The main 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

1. 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 modules that change their position relative to one another, thereby reshaping the system. A robotic system that changes its shape due to individual module motion has been called *self-reconfigurable*[5] or *metamorphic*.[2]

A self-reconfigurable robotic system is a collection of independently controlled, mobile modules, each of which has the ability to connect, disconnect, and move around adjacent modules. Metamorphic robotic systems, 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 between modules are as small as possible. In these systems, modules achieve locomotion by moving over a substrate composed of one or more other modules. The mechanics of locomotion depends on the hardware and can include module deformation to crawl over neighboring modules[3,9] or to expand and contract to slide over neighbors.[10] Alternatively, moving modules may be constrained to rigidly maintain their original shape, requiring them to roll over neighboring modules.[6,13,14]

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as bridge building, satellite recovery, or tumor excision.[9] The complete interchangeability of the modules 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 module motions required to go from a given initial configuration (I) to a desired goal configuration (G).

Many developers of self-reconfigurable robotic systems[5–7,9,10,12,13] 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.[1,3,5,9,10,12] Others use distributed approaches which rely on heuristic approximations and require communication between modules in each step of the reconfiguration process.[6,7,13,14]

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian.[3] 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 and the potential for greater system fault tolerance. Additionally, our strategy requires no communication between modules. We have previously applied this approach to the problem of reconfiguring a straight chain to an intersecting straight chain.[11]

Authors' email addresses are {jennyw,welch,amato}@cs.tamu.edu.

In this paper we address the problem of distributed reconfiguration from a straight chain of modules 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, finding a path with certain properties. The main result of the paper is a distributed algorithm for reconfiguring a straight chain into an admissible goal configuration, which uses the path found by the previous algorithm. Different heuristics for choosing the path are proposed to improve the performance of the reconfiguration algorithm and the performance of these heuristics is explored through simulation.[9]

2. RELATED WORK

Chirikjian[3] and Pamecha[9] discuss centralized algorithms for planar hexagonal modules that use the distance between all modules in I and the coordinates of each goal position to accomplish the reconfiguration of the system. Pamecha et al.[9] 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. Upper and lower bounds on the number of moves for reconfiguration between general shapes are given by Chirikjian.[3] The upper bound on the minimal number of moves is a function of the distance along the perimeter of the initial and final configurations, the maximum perimeter distance possible in a connected configuration of n modules, and the overlap between the initial and final configurations. Lower bounds for the general case are obtained by finding a perfect matching between modules in I and positions in G such that the sum of the distances between pairs is minimized.

Centralized motion planning strategies for systems of two dimensional robotic modules are examined by Nguyen et al.[8] and analysis is presented for the number of moves necessary for specific reconfigurations. The authors 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.

A centralized motion planning strategy for three dimensional cubic robots is presented by Rus and Vona.[10] In this paper, the proposed modules incorporate an actuator mechanism that causes module expansion and contraction, resulting in the sliding movement of a module over its neighbors. A centralized algorithm which takes $O(n^2)$ time to reconfigure a system of n modules is presented.

Centralized algorithms for decomposing a system of modules into a hierarchy of two dimensional substructures are presented by Casal and Yim.[1] Reconfiguration of the system involves connectivity changes within and between these substructures, along with substructure relocation. The paper concentrates on the decomposition algorithms and does not present algorithms for motion planning within substructures.

A distributed approach is taken by Murata et al. to reconfigure a system of two dimensional hexagonal modules,[6] and a system of three dimensional cubic modules.[7] In these approaches, each module senses its own connection type and classifies itself by the number of modules that 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. These distributed algorithms use random local motions to converge toward the goal configuration, a slow process that appears impractical for large configurations. These schemes also ignore the consequences of module collision and do not distinguish the relative location of modules in the plane, i.e., two configurations are the same if the modules composing them have the same connections.

Another distributed reconfiguration algorithm, for three dimensional rhombic dodecahedron shaped modules, is presented by Yim et al.[13] In this strategy, 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 distributed motion planning of three dimensional rhombic dodecahedral robots are presented by Zhang et al.[14] In this 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.

3. 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. We believe one contribution of our work is how our system model abstracts from specific hardware details about the modules.

In this paper, we consider two dimensional, hexagonal modules like those described by Chirikjian,[2] using his definition of lattice distance between modules in the plane. Our proposed scheme uses a new classification of module types based on connected edges similar to the classification used by Murata et al.[6] for connected vertices. In the algorithms presented in this paper, each module 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. Modules move from cell to cell and modify their state as they change position. Since the modules know the coordinates of the goal cells, we show that each of them can independently choose a motion plan that avoids module collision.

Because we are attempting to define the necessary building blocks for reconfiguration, the algorithms presented in this paper do not rely on communication between adjacent modules like the other distributed approaches.[6,13,14] One of our future goals is to determine how complex the configuration shapes can be before communication is required during reconfiguration.

In Sect. 4 we describe the system assumptions and the problem definition. Section 5 contains a centralized algorithm that determines whether or not a given configuration is admissible. Section 6 presents and analyzes a distributed algorithm for reconfiguring a straight chain to an admissible goal configuration. In Sect. 7 we present simulation results comparing the performance of our algorithm using different heuristics. Section 8 provides a discussion of our results and future work.

4. SYSTEM MODEL

4.1. Coordinate System

The plane is partitioned into equal-sized hexagonal cells and labeled using the coordinate system shown in Fig. 1, as in Chirikjian.[2]

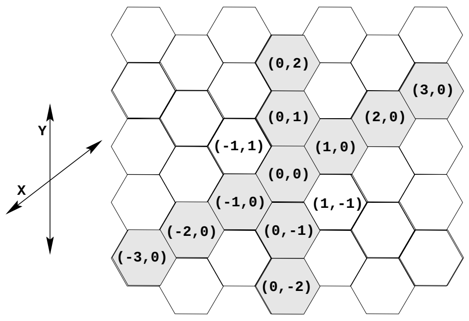

Figure 1. Coordinates in a system of hexagonal cells.

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.

4.2. 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. 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.

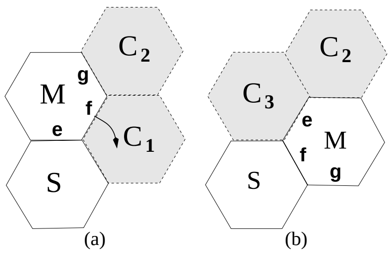

Figure 2. Before (a) and after (b) module movement: M is moving, S is substrate, C_1, C_2, and C_3 are empty cells.

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.

4.3. 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.

5. ADMISSIBLE CONFIGURATIONS

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

5.1. Definition of Admissible Configuration

Without loss of generality, assume I is 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, as shown in Figs. 3(a) and (b), where occupied cells have solid borders and goal cells are shaded. To see why these assumptions can be made without loss of generality, if I is a straight chain that is not oriented in this way, the algorithms we presented for straight chain to straight chain reconfiguration[11] can be used to reorient I in relation to 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 from west to east.

A *substrate path* p is a sequence of distinct cells, $c_1, c_2, \ldots, c_k$, such that

- each cell is adjacent to the previous, *but not to the west*,
- 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).

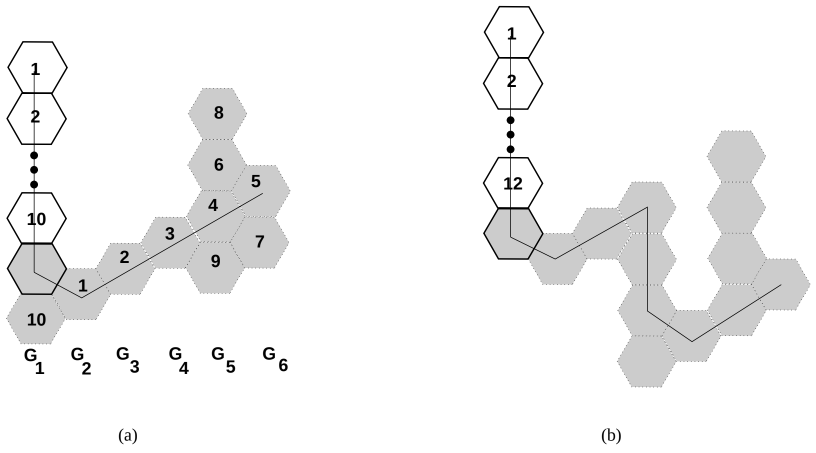

Figure 3. Example admissible (a) and inadmissible (b) G.

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

A substrate path is *admissible* if

- for each south segment of p ending with c_i, no goal cell is north of c_{i+1}, c_{i+2}, or c_{i+3}, and
- for each north segment of p ending with c_i, no goal cell is south of c_{i+1}, c_{i+2}, or c_{i+3}.

G is an *admissible* goal configuration if

1. each column G_i of G, $1 \leq i \leq m$, is contiguous and
2. there exists an admissible substrate path in G with respect to the cell north of the intersection of I and 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 correct. 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.[11] But ruling out any configuration in which modules have to move through an acute angle turn would be overly restrictive, since any two straight chains meet at either an acute or obtuse angle due to module shape. Since moving modules separated by two empty cells can move though acute angles without becoming deadlocked, we chose to base our definition of admissibility on configuration surfaces over which moving modules with two empty cells between them can move without becoming deadlocked.

Figure 3(a) depicts an example of an admissible configuration of G, where the line through I and G is an admissible substrate path. Figure 3(b) depicts a configuration of G that violates admissibility condition 2. The substrate path shown is inadmissible, as is every other possible substrate path for this configuration.

Our definition of admissible classes of goal configurations differs from that presented by Rus and Vona[10] because the modules used by these authors were 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.,[8] 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,[2] where locomotion is accomplished by a combined rigid body rotation and shape transformation produced by changing joint angles.

5.2. Detecting Admissible Configurations and Finding Substrate Paths

Condition 1 for determining the admissibility of G can be easily accomplished by scanning G in columns from north to south, northwest to southeast, and northeast to southwest, to determine if there exists an orientation in which each G_i is contiguous. If there is no orientation in which each G_i is contiguous, then G is not admissible.

Our procedure for finding an admissible substrate path in G (condition 2 for the admissibility of G) proceeds by first constructing a directed graph H as follows:

- Label the columns of G as described in Sect. 5.1, with the cells in each 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 position directly north of cell $G_{1,1}$ and call this node $G_{1,0}$. 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 $\mathrm{N}_{i,j}$, $\mathrm{S}_{i,j}$, $\mathrm{NE}_{i,j}$, and $\mathrm{SE}_{i,j}$, respectively (note that some of these cells might not be goal cells and thus are not represented in the graph).

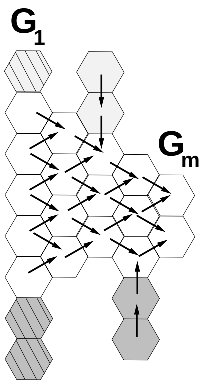

Figure 4. Directed graph H formed by algorithm.

The first phase directs edges in the undirected graph and marks the nodes that are determined to have an admissible path to a goal cell in the easternmost column. The columns are processed from east to west. First, every node in column G_m is marked. As shown in Fig. 4, each column west of column G_m consists of three segments: (A) the north segment of nodes with no goal cells to the east (shaded light gray), (B) the central segment of nodes that have goal cells to the east (unshaded), and (C) the south segment of nodes that have no goal cells to the east (shaded dark gray). Segment (A) is initially skipped. Each node in segment (B) is given an outgoing edge to each of its *marked* east neighbors. If the node has at least one such neighbor, then it is marked. Nodes in segment (C) 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 goal cells in a local neighborhood satisfy a certain “admissibility” condition (discussed below). Finally, nodes in segment (A) are processed south to north. Similarly to segment (C), each node is marked and given a directed edge to its south neighbor if the south neighbor is marked and if the goal cells in a local neighborhood satisfy a certain “admissibility” condition (discussed below). The arrows in Fig. 4 show the edges that are directed and the direction given to the edges and the cross-hatched cells are those that remain unmarked after the algorithm has been run. The cell on the north and the two cells on the south of column G_1 do not satisfy the “admissibility” condition, so no edges are directed from these cells in the algorithm.

The following pseudocode directs some of the edges in the graph, as described above. The variables used in the pseudocode are as follows:

- $onPath_{i,j}$: Boolean variable. Initially, $onPath_{i,j}$ is false for all goal cells in columns $1 \leq i \leq m-1$ and true for all nodes in column G_m. At a particular node i, the status of the $onPath_{i,j}$ variable at the nodes $\mathrm{N}_{i,j}$, $\mathrm{S}_{i,j}$, $\mathrm{NE}_{i,j}$, and $\mathrm{SE}_{i,j}$ is $onPath_{N_{i,j}}$, $onPath_{S_{i,j}}$, $onPath_{NE_{i,j}}$, and $onPath_{SE_{i,j}}$.
- x: Variable used to save the position of the southernmost cell that has not been checked by the algorithm.
- d: Direction to be checked, either $N = \bar{S}$ or vice versa.
- *remove*: Set of goal cell coordinates. Initially, $remove = \{\emptyset\}$ at all nodes.
- *path*: List of coordinates of goal cells that are added to the substrate path.

```
For each column $i$ := $m-1$ downto 1 do:
  $x$ := 1
  $j$ := 1
  while ($j \leq |G_i|$)
      while ($G_{i,j}$ has no adjacent nodes to the east)
          $j$++
      end while
      $x$ := $j-1$
      while (($j \leq |G_i|$) and
        ($G_{i,j}$ has $\geq$ one adjacent node to the east))
          if (($G_{i,j}$ has node to NE) and
            ($onPath_{NE_{i,j}}$)) then
              $onPath_{i,j}$ := true
              direct edge to NE
          end if
          if (($G_{i,j}$ has node to SE) and
            ($onPath_{SE_{i,j}}$)) then
              $onPath_{i,j}$ := true
              direct edge to SE
          end if
          $j$++
      end while
      while ($j \leq |G_i|$)
          if (($onPath_{N_{i,j}}$) and
            ($isAdmissible(N, i, j)$)) then
              $onPath_{i,j}$ := true
              direct edge to N
          end if
          $j$++
      end while
      while ($x > 0$)
          if (($onPath_{S_{i,x}}$) and
            ($isAdmissible(S, i, x)$)) then
              $onPath_{i,x}$ := true
              direct edge to S
          end if
          $x$--
      end while
  end while
```

```
Procedure $isAdmissible(d, i, j)$ returns boolean
    if ($\exists$ a goal cell in or $\bar{d}$ of: $dE_{i,j}$, $\bar{d}E$ of $dE_{i,j}$, or $\bar{d}E$ of $\bar{d}E$ of $dE_{i,j}$) then   // Case 1
        return false
    end if
    if (($i \leq m-4$) and (cell $dE$ of $d_{i,j}$ is goal cell)
      and (the cell $\bar{d}E$ of $\bar{d}E$ of $dE$ of $d_{i,j}$ is a goal cell)) then                // Case 2
        if (the cell $\bar{d}E$ of $dE$ of $d_{i,j}$ is not a goal cell)
            return false
        end if
        else if ($\exists$ an edge directed $\bar{d}$ out of the cell $\bar{d}E$ of $dE$ of $dE$ of $d_{i,j}$) then
            $remove_{i,j}$ = $\{$($dE$ of $dE$ of $d_{i,j}$), ($\bar{d}E$ of $dE$ of $dE$ of $d_{i,j}$$\}$
        end else if
    end if
    return true
```

From the *isAdmissible* procedure, we can see that if any edges at a node are directed to the east, then no edges at that node will be directed to the north or south. Also, since the cells in each column are contiguous, if an edge at a node is directed to the north, then no edge at that node will be directed to the south and vice versa.

After constructing H, if $onPath_{1,0}$ = true, then there exists an admissible substrate path from $G_{1,0}$ to some cell in G_m because of the way H is constructed. We believe, but have not yet proven, that if the algorithm fails to find an admissible substrate path with respect to $G_{1,0}$, then G does not contain such a path.

To find an admissible substrate path, we begin at node $G_{1,0}$ and move in any allowable direction (i.e., over any directed edge to a goal cell for which *onPath* is true) until reaching some goal cell in column G_m. If a node has a directed edge to only one neighbor for which *onPath* is true (either N, S, NE, or SE), then we go in that direction. The only other possibility is that a node has two neighbors for which *onPath* is true, NE and SE. In this case, a heuristic is used to decide whether to go NE or SE. If the decision to go N or S is taken, then certain nodes in the graph in columns to the east have *onPath* set to false (the "remove" cells calculated in the admissibility check, case 2). The choice of this edge means that certain later choices are no longer available.

The following algorithm is used to find and construct an admissible substrate path:

```
Initially, $i$ := 1 and $j$ := 0 and $path$ := $\langle G_{1,0} \rangle$
 1. while (($onPath_{i,j}$) and ($i < m$))
 2.     if $G_{i,j}$ has at least one east neighbor with $onPath$ true then
 3.          update $i$ and $j$ to index one of those neighbors (heuristic choice)
 4.          append $G_{i,j}$ to $path$
 5.     end if
 6.     else if $G_{i,j}$ has a north or south neighbor then
 7.         update $i$ and $j$ to index that neighbor   // will only have one
 8.         append $G_{i,j}$ to $path$
 9.         for all goal cells with coordinates in $remove_{i,j}$, set $onPath$ = false
10.     end else if
11. end while
```

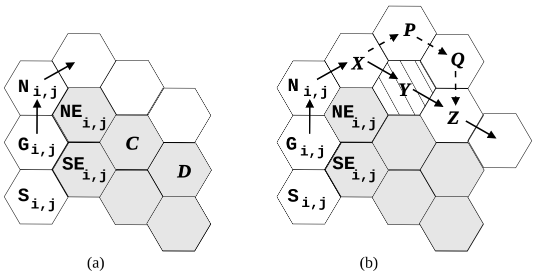

Figure 5. Scenarios for case 1 (a) and case 2 (b) of *isAdmissible* procedure.

The *isAdmissible* procedure is important both for constructing an admissible path from cell $G_{1,0}$ to some cell in column G_m and for dynamically choosing goal cells as they are included in an admissible substrate path. These functions are illustrated in Fig. 5, parts (a) and (b). In this figure, the unshaded cells are goal cells, the shaded cells are non-goal cells, the cross-hatched cell may or may not be a goal cell, and arrows indicate directed edges. The decision statement labeled case 1 in the *isAdmissible* procedure and depicted in Fig. 5(a), ensures that there are no goal cells in or south of the non-goal cells marked $NE_{i,j}$, C, and D, if the edge ($G_{i,j}$, $N_{i,j}$) is directed N and $onPath_{i,j}$ is set to true. If there are goal cells in or south of $NE_{i,j}$, C, or D, edge ($G_{i,j}$, $N_{i,j}$) will remain undirected and $onPath_{i,j}$ will remain false. An example of a case 1 violation is shown in Fig. 4, where the south edge out of cell $G_{1,0}$ is not directed in H, so $onPath_{1,0}$ remains false.

Figure 5(b) depicts the function of the decision statement labeled case 2 in the *isAdmissible* procedure for goal cells in column G_i, where $i \leq m - 4$. If the cells labeled X and Z in Fig. 5(b) are goal cells and cell Y is not a goal cell, then goal cell $G_{i,j}$ is not part of H. If the cells labeled X, Y, and Z are goal cells and goal cells $G_{i,j}$ and $N_{i,j}$ are added to the substrate path, then if cell Q has an edge directed to the south, the coordinates of goal cells P and Q are added to $remove_{i,j}$. Goal cells P and Q are then removed from possible inclusion in the admissible substrate path when goal cells $G_{i,j}$ and $N_{i,j}$ are included. This prevents the inadmissible dashed path through cells P, Q and Z from being added to the substrate path after cell $N_{i,j}$ is added.

The removal of goal cells P and Q from consideration for inclusion in an admissible substrate path will not prevent the algorithm from finding an admissible substrate path. To see why this is so, refer to Fig. 5(b) and consider that since goal cells $G_{i,j}$ and $N_{i,j}$ were added to H, goal cell Z cannot have a directed edge to the south, by case 1 of the *isAdmissible* procedure. Also, by case 2 of the *isAdmissible* procedure, goal cell Q has an edge directed to the south and no edges directed NE or SE. Thus, goal cell Z must have an edge directed to the SE or *onPath* would be false at Z. Therefore, the substrate path will continue to progress eastward through goal cell Z after goal cells P and Q are removed from consideration for inclusion in an admissible substrate path.

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.

6. DISTRIBUTED RECONFIGURATION ALGORITHM

6.1. 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.
5. I and G overlap in goal cell $G_{1,1}$, as described in Sect. 5.1.

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, beginning with the cell on the substrate path which has a directed edge incoming from cell $G_{1,1}$.
- The coordinates of cells in G that are north of the substrate path are stored in a list starting with the cell adjacent to and north of the cell on the substrate path in G_m to $G_{m,1}$, followed by the cell adjacent to and north of the cell on the substrate path in G_{m-1} to $G_{m-1,1}$, and so on, ending with the northwesternmost cell north of the substrate path in G.
- The coordinates of cells in G that are south of the substrate path are stored starting with the cell adjacent to and south of the cell on the substrate path in G_m to $G_{m,j}$, where j is the length of column G_m , followed by the cell adjacent to and south of the cell on the substrate path in G_{m-1} to $G_{m-1,k}$, where k is the length of column G_{m-1}, and so on, ending in the southwesternmost cell south of the substrate path in G.

6.2. Overview of Algorithm

The algorithm works in synchronous rounds. In each round, each module calculates whether it is free (cf. Fig. 6). In this figure, the modules labeled *trapped* are unable to move due to hardware constraints and those labeled *free* represent modules that must 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.

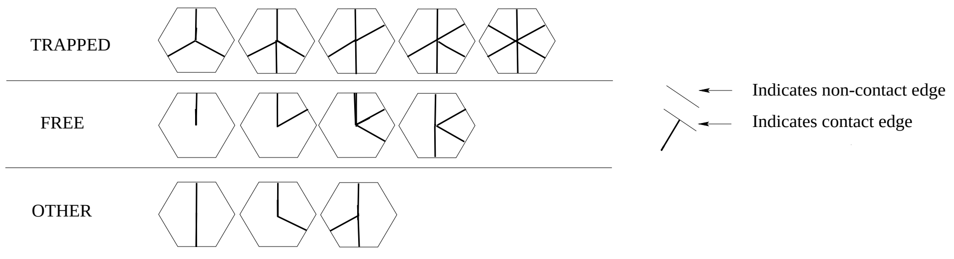

Figure 6. Contact patterns possible in algorithm.

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 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 3(a) has numbered goal cells showing how initial module positions correspond to final goal positions.

As in our previous paper,[11] modules use specific patterns of rotation and delay in our algorithm, as listed below. Note that only patterns 2 and 4 are used in our general algorithm schema.

1. *(0,0)-bidirectional*: modules alternate direction with no delay after free.
2. *(1,0)-bidirectional*: modules alternate direction with delay of 1 time unit after free for modules in positions > 1 rotating CW and no delay after free for modules rotating CCW.
3. *1-unidirectional*: modules rotate same direction with delay of 1 after free for modules in positions > 1.
4. *2-unidirectional*: modules rotate same direction with delay of 2 after free for modules in positions > 1.

The reconfiguration proceeds as follows:

- For modules in positions 1 through $|p|$:
  - Modules use *2-unidirectional* pattern in CW direction.
  - When a module is in the goal cell that it should occupy in p, it stops in that cell.
- 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.
  - 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.

6.3. Algorithm Pseudocode

The algorithm uses the following local variables at each module:

- n: Number of cells in G and number of modules in I.
- *myCoord*: The coordinates of the module in the plane.
- *position*: Order of modules in I, starting at the northernmost end of I. Initially calculated as $n - LD($*myCoord*, coordinates of $G_{1,1})$.
- d: Variable containing the direction of movement, CW or CCW.
- *delay*: Number of time units module waits after it is free and before it makes its first move. Initially set to 0.
- *myGoalCoord*: Coordinates of goal cell in which module will stop moving. Initially module in I overlapping G has *myGoalCoord* = coordinates of $G_{1,1}$ and all other modules in I calculate *myGoalCoord* after calculating their *position*.
- *substrateCoords*, *coordsNorth*, and *coordsSouth*: Arrays of coordinates of goal cells on, north, and south of the substrate path, in the order described in Sect. 6.1.

```
Code for each module for which $myCoord \neq myGoalCoord$:
  Initially:
     Modules calculate $position$ based on distance from $G_{1,1}$,
       then calculate $myGoalCoord$, rotation direction $d$, and $delay$
       based on the value of $position$ and the length of the $substrateCoords$,
       $coordsNorth$, and $coordsSouth$ arrays.
```

```
In round $r$ := $1, 2, \ldots$ :
  if ($isFree$())
      if ($delay$ = 0) then
          move $d$
      end if
      else
          $delay$--
      end else if
  end if

Procedure $isFree$():
   Return true if contact pattern is $free$ (cf. Fig. 6) and false otherwise
```

7. SIMULATION RESULTS

Our simulation experiments were inspired by the work of Pamecha et al.,[9] 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 experimented with running our algorithm on various shapes using different numbers of modules. In these preliminary simulation results, we tested the effect of the heuristic choice in line 3 of the algorithm to find an admissible substrate path (presented in Sect. 5.2) on the performance of the reconfiguration algorithm, where 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 7 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.

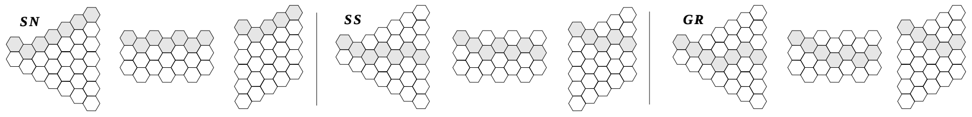

Figure 7. Example paths found for SN, SS, and GR heuristics.

In Figs. 8 and 9(a), we depict the results obtained from experiments with wedges of similar orientation and increasing size. Figures 8 and 9(b) show the results obtained when experiments were performed on lengthening rectangles and Figs. 8 and 9(c) show the results on diamond shapes. For each shape, the number of moves increased more than linearly for increasing values of n and the number of moves was nearly the same for each heuristic for all values of n. For each shape 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 directed path of marked nodes may be chosen as the substrate path, heuristics can improve the number of rounds required for reconfiguration.

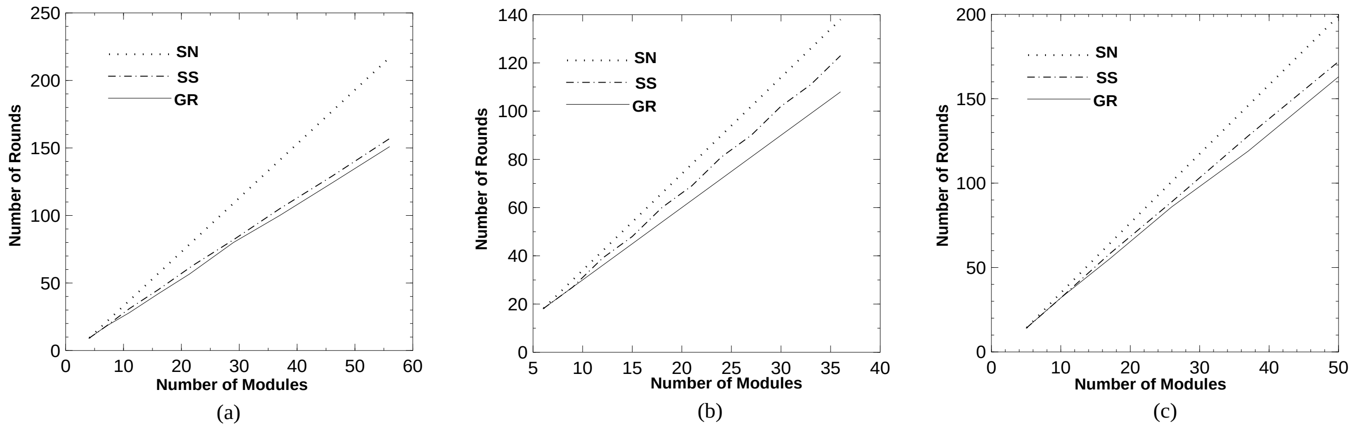

Figure 8. Rounds used for wedge (a), lengthening rectangle (b), and diamond shaped (c) configurations.

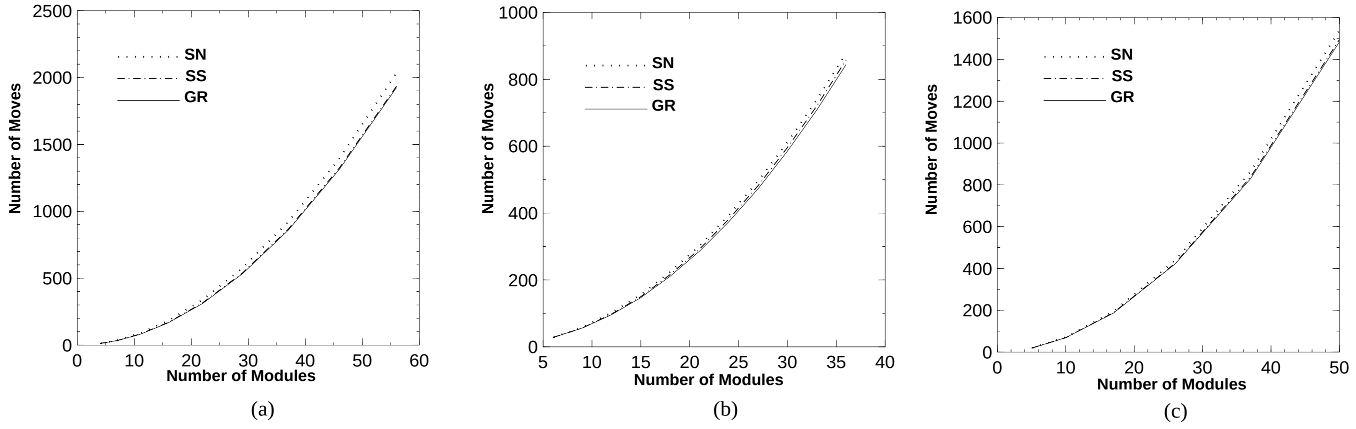

Figure 9. Moves used for wedge (a), lengthening rectangle (b), and diamond shaped (c) configurations.

8. CONCLUSIONS AND FUTURE WORK

The algorithm presented in this paper relies on total knowledge of the goal configuration. Each module precomputes all aspects of its movement once it has sufficient information to reconstruct the entire initial configuration. 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 them. We have some initial ideas for ways to deal with module collision and deadlock on the fly, which we are currently testing and refining.

We are experimenting with different heuristics to improve the time used for 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.[11] Such a heuristic improvement makes the initial determination of final goal position slightly more complex, however, since modules do not always arrive on the substrate path in the order of their initial positions in I. Another heuristic improvement is to choose a substrate path for which all modules on the north or south of the path meet the path at an obtuse angle, since then the inter-module spacing can be reduced to one space.

ACKNOWLEDGMENTS

We thank Debra Elkins for her assistance with the graphs in Sect. 7. Walter is supported by GE Faculty of the Future and Department of Education GAANN fellowships. Welch is supported in part by NSF Grant CCR-9972235. Amato is supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, EIA-9805823, and

EIA-9810937, DOE ASCI ASAP (Level 2 Program) grant B347886, and by the Texas Higher Education Coordinating Board under grant ARP-036327-017.

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. G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 449–455, 1994.
3. G. Chirikjian and A. Pamecha. Bounds for self-reconfiguration of metamorphic robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1452–1457, 1996.
4. K. Kotay and D. Rus. Motion synthesis for the self-reconfiguring molecule. In *IEEE Intl. Conf. on Robotics and Automation*, pages 843–851, 1998.
5. 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.
6. S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 441–448, 1994.
7. 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.
8. A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. To appear in *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.
9. 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.
10. D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 2513–2520, 1999.
11. 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.
12. M. Yim. A reconfigurable modular robot with many modes of locomotion. In *Proc. of Intl. Conf. on Advanced Mechatronics*, pages 283–288, 1993.
13. M. Yim, J. Lamping, E. Mao, and J. G. Chase. Rhombic dodecahedron shape for self-assembling robots. SPL TechReport P9710777, Xerox PARC, 1997.
14. Y. Zhang, M. Yim, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. To appear in *Autonomous Robots Journal, special issue on self-reconfigurable robots*, 2000.