# Choosing Good Paths for Fast Distributed Reconfiguration of 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) from specific initial to specific goal configurations. The initial configuration we consider is a straight chain of modules, while the goal configuration satisfies a simple admissibility condition. Reconfiguration of the modules 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 examine the problem of finding the substrate path most likely to result in fast parallel reconfiguration, drawing on results from our previous papers [12, 13, 14]. Admissible goal configurations are represented as directed acyclic graphs (DAGs). We present a combination graph traversal-weighting algorithm that traverses all paths in the rooted DAG and use this algorithm to determine the best substrate path. We extend our definition of admissible substrate paths to consider admissible obstacle surfaces for reconfiguration when obstacles are present in the environment.*

## 1 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 [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 between adjacent modules are as small as possible. 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 to crawl over neighboring modules [4, 10] or to expand and contract to slide over neighbors [11]. Alternatively, moving robots may be constrained to rigidly maintain their original shape, requiring them to roll over neighboring robots [7, 16, 17].

*This research supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874.

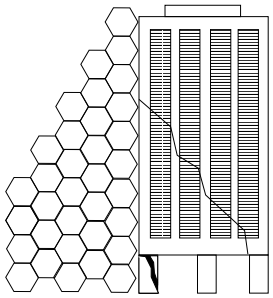

Figure 1: Metamorphic robots used to buttress a building.

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as bridge building, structural support, satellite recovery, or tumor excision [10]. 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 ($G$). Most of the existing motion planning strategies rely on centralized algorithms to plan and supervise the motion of the system components [4, 6, 10, 11, 15]. Others use distributed approaches which rely on heuristic approximations or require communication between robots in each step of the reconfiguration process [1, 7, 8, 16, 17].

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian [4]. 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 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 [13] and a straight chain to a goal configuration that satifies a general "admissibility" condition [12, 14]. In these papers, a centralized algorithm was described for determining whether an arbitrary goal configuration is admissible.

This paper presents an algorithm to rank candidate sub-

strate paths in an admissible goal configuration, allowing flexibility in choosing the location of $I$ based on that substrate path. This flexibility in choosing the intersection point of $I$ and $G$ allows our reconfiguration algorithms to be applicable in many more scenarios than in our previous work. Another contribution of this paper is the adaptation of the distributed reconfiguration algorithm presented in [14] to provide better parallelism based on the choice of substrate path. Lastly, we introduce our new work on reconfiguration when obstacles are present in the environment. Our admissibility criteria for a substrate path can be readily extended to reconfiguration in the presence of obstacles.

# 2 Related work

Chirikjian [4] and Pamecha [10] 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. [10] 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.

Centralized motion planning strategies for systems of two dimensional robotic modules are also examined by Nguyen et al. [9] and analysis is presented for the number of moves necessary for specific reconfigurations.

A centralized motion planning strategy for three dimensional cubic robots is presented by Rus and Vona [11]. A set of distributed motion planning algorithms for a system of cubic robots is presented by Butler et al. in [1]. In another paper [2], Butler et al. present a rule set that can be run by vertical "layers" of cubic modules and a distributed control algorithm for locomotion is described that will work in any system composed of cubic modules. This paper also presents a rule set for distributed control of cubic modules when obstacles are present in the environment.

Distributed approaches are taken by Murata, et al. to reconfigure a system of two dimensional hexagonal modules [7], and a system of three dimensional cubic modules [8]. Yim et al. [16] and Zhang et al. [17] present distributed algorithms to reconfigure three dimensional rhombic docecahedral modules. Each of these algorithms are probabilistic and require substantial message passing between neighboring modules.

**Our approach**

This paper examines 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 [12, 13, 14], can be accomplished using algorithms that do not require any message passing. Therefore, our algorithms are more communication efficient than the distributed approaches of [1, 7, 16] and [17].

In this paper, we consider two dimensional, hexagonal robots like those described by Chirikjian [3]. Our proposed scheme uses a classification of robot types based on connected edges similar to the classification used by Murata et al. [7] 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.

In Section 3 we describe the system assumptions and the problem definition. Section 4 describes our algorithm for determining admissibility of a goal configuration and presents a new graph traversal and weighting algorithm for planning the reconfiguration. Section 5 presents a distributed algorithm for reconfiguring a straight chain to an admissible goal configuration. Section 6 introduces admissibility conditions for obstacles and suggests a method for reconfiguration in the presence of obstacles. Section 7 provides a discussion of our results and future work.

# 3 System model

**Assumptions about modules**

The plane is partitioned into equal-sized hexagonal cells and labeled using the same coordinate system as described by Chirikjian [3].

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.

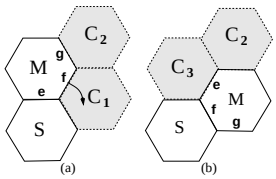


Figure 2: Before (a) and after (b) module movement: $M$ is moving, $S$ is substrate, and $C_1$, $C_2$, and $C_3$ are empty cells.

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.

**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.

# 4 Admissible 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*. Informally, 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 or collide.

**Admissibility definitions**
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 $n$. Figure 3 gives examples of orientations of $I$ and $G$ that satisfy these assumptions in which $n = 6$. In this figure, cells in $I$ are numbered with solid borders and goal cells are shaded.

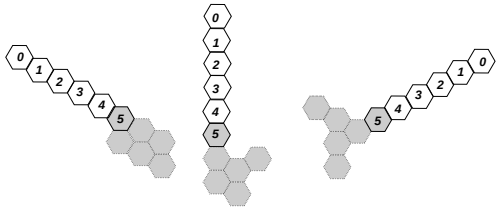


Figure 3: Example orientations of $I$ and $G$.

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 $G_1$. Without loss of generality, suppose that $G$ is oriented such that column $G_1$ is the westernmost column, $G_m$ is the easternmost column, and each column of $G$ is a contiguous straight chain oriented north-south. Figure 5 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 [13] for straight chain to straight chain reconfiguration can be used to reorient $I$ in relation to $G$.

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

**Definition 1** *A* **segment** *of $p$ is a contiguous subsequence of $p$ of length $\geq 2$. In a* **south segment**, *each cell is south of the previous and analogously for a* **north segment**.

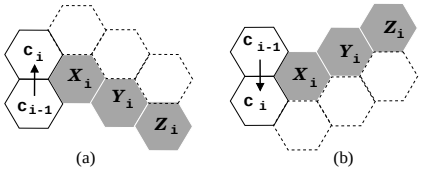


Figure 4: Labels for north segment ending in $c_i$ (a) and south segment ending in $c_i$ (b) (cells that must not be goal cells are shaded).

**Definition 2** *$p$ is an* **admissible path** *if*

1. *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) *the cells labeled $X_i$, $Y_i$, and $Z_i$ in Figure 4(a) are not goal cells and*
   (b) *$c_{i+1}$, $c_{i+2}$, and $c_{i+3}$ do not form any south segments, and*
3. *for each south segment of $p$ ending with $c_i$,*
   (a) *the cells labeled $X_i$, $Y_i$, and $Z_i$ in Figure 4(b) are not goal cells and*
   (b) *$c_{i+1}$, $c_{i+2}$, and $c_{i+3}$ 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. Segments directed to the east may be referred to as *horizontal* segments when specific direction is not important.

**Definition 3** *$p$ is a* **substrate path** *if*

- *$p$ begins with the cell in which $I$ and $G$ overlap,*
- *subsequent cells are all in $G$, and*
- *$p$ spans $G$, from column $G_1$ to column $G_m$.*

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

The admissibility conditions for a substrate path are directly related to the degree of parallelism possible, 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 [13]. 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.

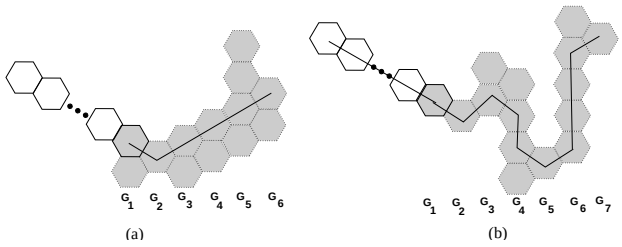


Figure 5: Example admissible (a) and inadmissible (b) $G$ (cells in $I$ have solid borders and cells in $G$ are shaded).

Figure 5 depicts an example of an admissible (a) and an inadmissible (b) configuration of $G$.

**Finding substrate paths**
Our procedure for finding an admissible substrate path in $G$ proceeds in three steps:

1. Construct a directed graph $H$ from $G$.
2. Weight the vertices in $H$ and calculate the cost of all possible directed paths from each cell in $G_1$ to every cell in column $G_m$. Do this for all orientations of $G$ for which columns $G_1 \ldots G_m$ are contiguous.
3. Determine which paths in $H$ have lowest cost and most evenly bisect $G$. Select a substrate path and intersection for $I$ based on these criteria.

Step 1 is done as described in [14] and is only reviewed briefly in this paper. Steps 2 and 3 have not been described previously.

**Constructing $H$**
The graph $H$ is initialized as follows:

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

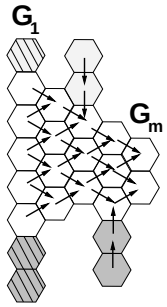


Figure 6: Directed graph $H$ formed by algorithm.

The columns of $G$ are processed from east to west. First, every node in column $G_m$ is marked. As shown in Fig. 6, 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) of each column is initially skipped. Each node in segment (B) is given an outgoing edge to each of its *marked* east neighbors, with the exception of the situation where 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. 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 the local neighborhood satisfy the admissibility conditions for that edge to be included in a substrate path. Finally, nodes in segment (A) are processed south to north. Each node is marked and given a directed edge to its south neighbor in a manner analogous to the nodes in segment (C).

The arrows in Fig. 6 show the edges that are directed and the direction given to the edges. The cross-hatched cells are those that remain unmarked after the algorithm has been run. The full pseudocode for this algorithm can be found in [14]. We proved in that paper that the action of this algorithm ensures that no inadmissible substrate paths will be produced from directing edges in $H$.

**Traversing and weighting $H$**

We combine a weighting scheme with a graph traversal algorithm for the purposes of assigning a weight to each path that spans all columns of $H$. We describe the traversal algorithm first, then the weighting scheme. As previously mentioned, our technique traverses all potential substrate paths, and thus provides a general technique for traversing all root to leaf paths in a rooted DAG.

The *TraverseGraph* algorithm proceeds as follows:

- Initially, all vertices in $H$ are white (unvisited).
- Let $v$ be a marked cell in column $G_1$. Then $v$ is the root of a DAG in $H$. Colour $v$ black (visited).
- While $v$ has white (unvisited) children, choose a child, $c$, and mark $v$ as the parent of $c$.
- Colour $c$ black and continue traversing from $c$.
- If $v$ is a leaf and $v$'s parent has a white (unvisited) child,
  - then back up to $v$'s parent and continue traversing from there.
  - else, if $v$ is a leaf and $v$'s parent has no white (unvisited) children, color $v$ and $v$'s sibling white (i.e., unvisit them) and back up to $v$'s parent. Continue backtracking from there until reaching the root.

The pseudocode for *TraverseGraph* and its internal procedure *Backtrack* is presented in Figure 7.

```
Procedure TraverseGraph(vertex v)
Initially, all v ∈ H are white (unvisited) and parent_v = ∅.
Let v = root of a DAG in H starting in column G_1
 1.    color v black (visited)
 2.    if v has a white (unvisited) child
 3.         pick a child, c
 4.         parent_c := v
 5.         TraverseGraph(c)
 6.    else if parent_v ≠ null //v is not the root
 7.         Backtrack(v)
 8.    end if
```

(a)

```
Procedure Backtrack(vertex v)
 1.    if parent_v has an unvisited (white) child
 2.         TraverseGraph(parent_v)
 3.    else if parent_v has no unvisited (white) children
 4.         backtrackParent := parent_v
 5.         color v and sibling_v white //unvisit them
 6.         set parent_v and parent_sibling_v to null
 7.         Backtrack(backtrackParent)
 8.    end if
```

(b)

Figure 7: Pseudocode for Procedures (a) *TraverseGraph* and (b) *Backtrack*.

Figure 8 shows an example of a graph traversal. In Figure 8(a), the root of the graph, $A$ is colored black (visited), while all other vertices remain white (unvisited). In Figure 8(b), a path from $A$ to the leaf $G$ has been traversed and all vertices along the path are colored black. In Figure 8(c), the algorithm backtracks to vertex $B$. Vertices $E$ and $G$ are colored white (they are "unvisited") since their respective parents have no unvisited children. $C$ remains black, however, since vertex $D$ has not been visited. In Figure 8(d), a path picking up at vertex $B$ and continuing to leaf $G$ is traversed. All vertices on the path are colored black. In Figure 8(e), the algorithm backtracks to vertex $D$. $G$ is colored white while $E$ remains black since $F$ has not been visited. In Figure 8(f), traversing continues from vertex $D$ to the leaf $G$, completing the last untraversed path in the graph.

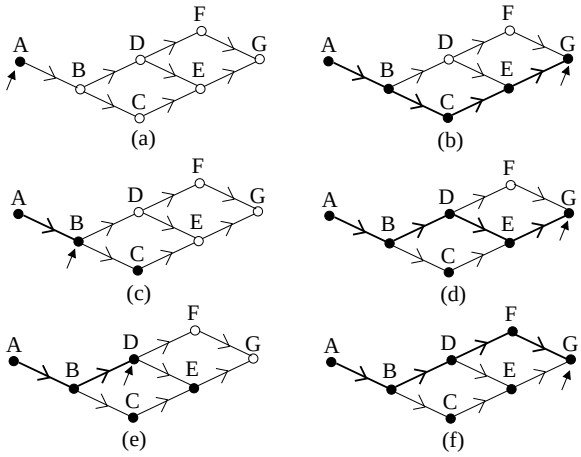


Figure 8: Example graph traversal. Darker lines indicate paths currently being traversed and small pointer indicates node currently being visited.

To find all possible substrate paths in $H$, the graph traversal algorithm is run once for each cell in column 1 of $H$ that has an outgoing edge (i.e., once with each cell in column $G_1$ as the root). During each root to leaf walk, a vertex receives a weight specified by the weighting scheme, described below.

We know from our previous work in [13] that straight chains of modules in $I$ can fill in collinear straight chains or chains with single obtuse angle bends faster than they can fill in chains with acute angle bends. For these collinear or single bend goal configurations, we showed that the reconfiguration can be done in optimal time because modules in $I$ can initially alternate rotation directions and move from the non-intersecting end of $I$ without delay. Thus, to maximize parallelism, we designed our weighting scheme to give the lowest weight to straight or single bend substrate paths that proceed horizontally across the columns of $G$.

We assign a separate weight value to each marked vertex in $H$ based on the direction of its incoming edge and that of its parent's incoming edge as follows:

- If a vertex has a vertical incoming edge, it has weight 10.
- Else if a vertex's incoming edge is directed in a different direction than its parent's incoming edge, it has weight 1.
- Else if a vertex's incoming edge is directed in the same direction as its parent's incoming edge, it has weight 0.

All nodes in column $G_1$ that have an outgoing edge are assigned weight 0.

The weight at each vertex on a directed path is summed with the weights of its ancestors, creating a "cumulative

weight" for the vertex that represents the cost of the path to that point. The weight of any leaf in $H$ represents the total cost of the path from the root to that leaf. The cost of each path is stored whenever a leaf is visited. Vertices are unweighted during the backtracking phase of the algorithm to ensure that a vertex's weight is based on the correct parent for each new path.

Vertices in columns numbered higher than 1 with incoming edges directed to the N or S are the most heavily weighted in our algorithm because vertical edges always form substrate paths with at least two bends in these cases. When neighboring modules in $I$ alternate rotation directions to fill a "multiple bend" substrate path, a precise sequence of initial module delays must be used to ensure that modules do not collide on the substrate path. Therefore, if a substrate path has multiple bends, we require that the modules in $I$ that will fill the substrate path all rotate the same direction, thereby sacificing parallelism in order to avoid collision.

It is clear that only paths formed by a straight, non-vertical chain of modules will have a total cost of 0. Likewise, only paths with a single NE or SE bend and no vertical segments will have a cost of 1. Figure 8(b) shows an example of a cost 1 path. Since paths with one or fewer obtuse angle bends can be filled most efficiently in terms of number of rounds used, the paths of cost 0 and 1 are preferable for selection as a substrate path. From our work in [14], we know that substrate paths that bisect the goal allow us to achieve the highest degree of parallelism, as they permit modules to fill in the goal bidirectionally. Paths of cost 0 and cost 1 are therefore processed before all other paths to determine which path bisects the goal configuration most evenly.

In the event that a cost 0 or 1 path does not come within one module of bisecting $G$, higher cost paths are considered. As before, higher cost paths that split the goal equally or almost equally are considered for selection first.

In the full paper, we prove that the *TraverseGraph* algorithm traverses every path from $G_1$ to $G_m$ in the graph $H$.

# 5 Distributed reconfiguration

In this section, we describe the distributed algorithm that performs the reconfiguration of $I$ to $G$ after an admissible substrate path is found using the algorithms described above.

**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 configuration.
4. $I$ and $G$ overlap in one goal cell in column $G_1$, as described in Sect. 4.

**Overview of algorithm**

The algorithm works in synchronous rounds. In each round, each module determines whether it is free (cf. Fig. 9). 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.

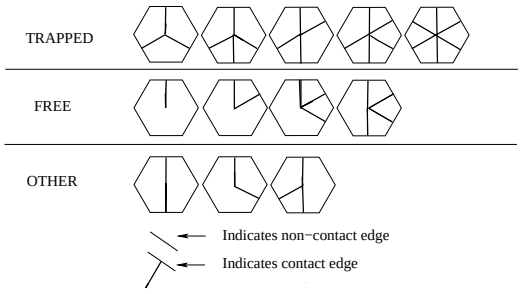


Figure 9: Contact patterns possible in algorithm.

Only module 0 (the module at the free end of $I$) can initially determine the exact time when it will begin moving. Other modules in $I$ rely on local contact information to calculate their position in $I$ and any possible delay after they become *free* to avoid collision and deadlock. Once a module begins moving, it has only the local information about contacts with adjacent modules and its current coordinates to guide its part of the entire system reconfiguration.

All modules except module 0 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, noting the direction (CW or CCW) in which the passing modules rotate. The module intersecting $G$ does not move.

Let $p$ be the array of coordinates of goal cells on the substrate path (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 $G_1$. 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 substrate path, and the current count of modules that have passed on both sides.

Modules fill in the substrate path first. Different patterns of delay and rotation are selected, depending on whether the cumulative cost of the substrate path is 0, 1, or greater than 1. After every goal cell in $p$ is filled, modules alternate rotation directions, filling the columns projecting north and south of $p$ from east, $G_m$, to west, $G_1$.

Modules use specific patterns of rotation and delay, as listed below.

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. *unidirectional*: modules rotate same direction with delay of 2 after free for modules in positions $> 1$.

The reconfiguration schema uses the cost of the substrate path found in the previous section and proceeds as follows:

- For modules 0 through $|p|$ (i.e., those modules filling in the substrate path):
  - If cost = 0, modules 0 through $|p|-1$ use *(0,0)-bidirectional* pattern, with module 0 starting in CW direction. Module $|p|$ begins the *(0,1)-bidirectional* pattern, moving in the opposite direction from module $|p|-1$ with delay of 2 (unless there are no cells to be filled in the opposite direction, in which case it begins the *unidirectional* pattern).
  - If cost = 1, modules use *(0,0)-bidirectional* pattern. If the distance from module 0 to the "bend" is odd, module 0 begins moving CCW, otherwise it begins moving CW. Modules enter the part of the substrate path before the bend in the same order they begin moving. Because of the bend in the substrate path, modules arrive on the "tail" of the substrate path in a different order than the order in which they begin moving. Let $i$ be the first module to choose a goal position in the part of the substrate path after the bend, i.e., the "tail" of the substrate path. If $n$ is even, modules arrive at their positions in the tail of the substrate path in this order: $i, i+2, i+1, i+4, i+3, i+6, i+5 \ldots$. Otherwise, if $n$ is odd, modules arrive at their positions in the tail of the substrate path in this order: $i+1, i, i+3, i+2, i+5, i+4, \ldots$. The module in position $|p|$ goes on the substrate path and the module in position $|p|-1$ does not when $n$ and the "tail" of the substrate path have different parity.
  - If cost $> 1$, modules $0 \ldots |p|-1$ use the *unidirectional* pattern in CW direction. Module $|p|$ begins *(0,1)-bidirectional* pattern, moving CCW (unless there are no cells to be filled in the CCW direction, in which case it continues the *unidirectional* pattern).
  - Each module stops in the goal cell on the substrate path that it has calculated it should occupy.
- For modules in positions $> |p|$ (i.e., these modules climb over the substrate path to fill the rest of $G$):
  - Modules use *(1,0)-bidirectional* pattern until all cells either north or south of $p$ are filled. After this, modules use *unidirectional* pattern, with either CW or CCW direction.
  - Each module stops in the goal cell to the north or south of the substrate path that it has calculated it should occupy.
- Once a module stops for a round in a goal cell, it never moves out of that goal cell.

The pseudocode used by all *free* modules during each round of the reconfiguration is shown in Figure 10. Local variables at each module include:

- *contacts*: Boolean array indicating on which edges a module has neighboring modules. Assumed to be automatically updated at each round by some lower layer.
- *position*: Order of modules in $I$, starting at the end of $I$ that is furthest from $G$. If the module is initially at distance $n-2$ from $G$, *position*$= 0$, otherwise position is calculated by counting passing modules.
- $d$: Direction of movement, CW or CCW.
- *flips*: Counter used to determine whether the module is free.
- *delay*: Number of time units module waits after it is free and before it makes its first move. Initially 0.

```
In round r := 1, 2, ... :
 1. if ((position = 0) or (IsFree()))
 2.    if (delay = 0)
 3.       move d
 4.    end if
 5. else
 6.    delay := delay−1
 7.    Count modules passing in CW and CCW directions
 8. end if

Procedure IsFree():
 1. flips := 0
 2. for (i := 0 to 5) do
 3.    if (contacts[i] ≠ contacts[(i + 1) % 6])
 4.       flips++
 5.    end if
 6. end for
 7. return ((position - 1 is unoccupied) and
       (flips = 2) and (number of contact edges < 5))
```

Figure 10: Pseudocode for all modules from straight chain to admissible $G$.

Each module calculates its rotation direction, delay before moving, and final goal coordinates after it determines its position in $I$. Modules in their initial positions keep separate tallies of other modules passing on the CW and CCW side.

For configurations with substrate path cost 0 or cost greater than 1, the calculation of final goal position is straightforward, since modules arrive in their calculated goal cells sequentially in the order they begin moving. Because of this sequential arrival pattern, modules in higher initial positions have an accurate view of the destination for each module that passed.

For configurations with substrate path cost 1, modules need to adjust the count of modules passing in the CW and CCW directions when the parity of $n$ and the "tail" of the substrate path is different. This is because modules arrive in the "tail" out of order, i.e., the module in position $|p|$ that modules with higher initial positions count as heading for a goal position north or south of the substrate path actually ends up in a position on the path. The pattern is predictable and therefore easily computed locally at each module.

# 6 Obstacles

In this section we consider the presence of obstacles in the coordinate system and present our preliminary ideas on reconfiguration in the presence of obstacles.

An obstacle is a sequence of one or more "forbidden cells" that modules cannot enter. Modules may, however,

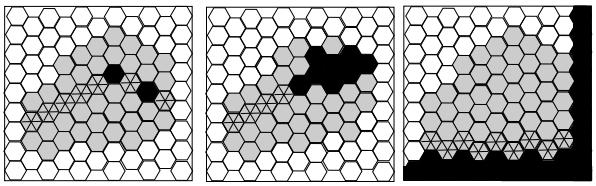

Figure 11: Admissible obstacles (shown in black) in three goal scenarios. Goal cells are shaded dark gray and those on the substrate path are starred.

touch obstacles and may use them as a substrate for movement. We informally define an obstacle as having an *admissible surface* if the perimeter of the obstacle is an admissible path. We then require that either 1) all "obstacle surfaces" adjacent to goal cells are admissible surfaces (when considering interaction with $G$ and obstacles), or 2) that an admissible surface(s) can be formed by "concatenating" the obstacles with modules. Obstacles may occur at any location in or around the goal. They may not, however, separate $I$ from $G$ by completely enveloping $G$.

Examples of admissible obstacles are shown in Figure 11, where the forbidden cells are black, the goal cells are gray, and the goal cells on the substrate path are marked with an asterisk. In each scenario of this figure, the substrate path has incorporated the admissible surface to form an admissible substrate path.

Using this definition of admissible surfaces, we intend to cope with the presence of obstacles both inside, adjacent to, and around $G$ by first analyzing the admissibility of the combination of obstacles, $G$, and $I$. We will choose a substrate path and location for $I$ that may include obstacles, and then select the substrate path such that it includes the admissible surfaces of obstacles if necessary. Modules will then move normally across the surface of obstacles during reconfiguration. For example, in Figure 12, the ground is treated as an admissible obstacle when a buttress is needed to hold up a skyscraper.

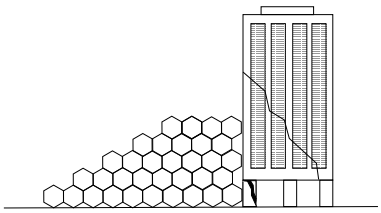

Figure 12: Obstacle surface (ground) used as foundation for reconfiguration.

# 7 Conclusions and future work

We have presented an algorithm for determining the substrate path that permits flexibility in choosing a point of intersection between the initial configuration and the goal and allows for maximum efficiency in reconfiguration. We also considered reconfiguration in the presence of obstacles, naturally extending our definition of admissible substrate paths to include obstacle surfaces.

We believe that this 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.

# References


[1] Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. In *Proc. of IROS 2001*, to appear.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *Proc. of the ICRA 2001 Workshop on Modular Robots*, 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 and A. Pamecha. Bounds for self-reconfiguration of metamorphic robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1452–1457, 1996.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] J. Walter, J. Welch, and N. Amato. Distributed reconfiguration of hexagonal metamorphic robots in two dimensions, in Sensor Fusion and Decentralized Control in Robotic Systems III, Gerard T. McKee and Paul S. Schenker, eds., *Proceedings of SPIE*, Vol. 4196, pp. 441-453, 2000.

[13] 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.

[14] J. Walter, J. Welch, and N. Amato. Distributed reconfiguration of metamorphic robots. Submitted, 2001.

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

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

[17] 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.