

Distributed Reconfiguration of Metamorphic Robot Chains

Jennifer E. Walter^{*}

Jennifer L. Welch[†]

Nancy M. Amato[‡]

ABSTRACT

The problem we address 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. We present a distributed algorithm for reconfiguring a straight chain of hexagonal modules at one location to any intersecting straight chain configuration at some other location in the plane. We prove our algorithm is correct, and show that it is either optimal or asymptotically optimal in the number of moves and asymptotically optimal in the time required for parallel reconfiguration. We then consider the distributed reconfiguration of straight chains of modules to a more general class of goal configurations.

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 inde-

^{*}Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. jennyw@cs.tamu.edu. Walter is supported by GE Faculty of the Future and Department of Education GAANN fellowships.

[†]Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. welch@cs.tamu.edu. Welch is supported in part by NSF Grant CCR-9972235.

[‡]Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. amato@cs.tamu.edu. 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.

pendently 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 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, 12, 13].

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, 6, 7, 9, 10, 11, 12] 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, 11]. Others use distributed approaches which rely on heuristic approximations and require communication between modules in each step of the reconfiguration process [6, 7, 12, 13].

We believe there is a rich opportunity for the distributed computing community to make a contribution in this problem domain by designing robust and efficient distributed motion planning algorithms for self-reconfiguration of metamorphic robotic systems. As a first step, 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 focus on a system composed of planar, hexagonal robotic modules as described in [3] as a starting point in this distributed motion planning.

2. RELATED WORK

The papers of [3] and [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. In [9], the distance between configurations is defined as a metric. This metric is applied 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 in [3]. The upper bounds on the minimal number of moves are functions 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. General lower bounds 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.

In [8], centralized motion planning strategies for systems of two dimensional robotic modules are examined 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. These authors define specific motion constraints based on the rigid nature of the modules and use knowledge about the initial configuration to plan the reconfiguration process.

Centralized motion planning algorithms are proposed in [5] and [4] that allow three dimensional modules to accomplish self-locomotion relative to a structure composed of identical robotic modules by pivoting on attachment points. These modules, called “molecules”, are composed of identical units connected by a rigid bond.

A centralized motion planning strategy for three dimensional cubic robots is presented in [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.

Centralized algorithms for decomposing a system of modules into a hierarchy of two dimensional substructures are presented in [1]. Reconfiguration of the system then involves connectivity changes within and between these substructures, along with substructure relocation. The algorithms for motion planning within substructures are not presented.

A distributed approach is taken in [6] to reconfigure a system of two dimensional hexagonal modules, where each module senses its own connection type and classifies itself by the number of modules that it physically contacts. Since contacts are limited to the vertices of a planar hexagonal module, the number of connection types is finite and a subset of

these types are designated *movable*. 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.

In [7], a distributed reconfiguration algorithm for three dimensional cubic modules is presented. The distributed approaches in [6] and [7] 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 in [12]. 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 to heuristically choose moves that lower the distance to the goal configuration.

In [13], several heuristic approximation algorithms for distributed motion planning of three dimensional rhombic dodecahedral robots are presented. 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.

Our approach: This paper will examine distributed motion planning strategies for a planar metamorphic robotic system undergoing a very simple reconfiguration, from a straight chain to any intersecting straight chain. 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 in [2], using their 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 [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, each of them can independently choose a motion plan that avoids module collision.

Unlike the authors of [6, 12] and [13], we have chosen to develop the distributed reconfiguration algorithm starting from a simple, rather than a complex, initial configuration, in the attempt to rigorously define the code and data structures necessary at each module. Because we are attempting to define the necessary building blocks for reconfigu-

ration, the algorithms presented in this paper do not rely on communication between adjacent modules like the other distributed approaches. One of our future goals is to determine how complex the configuration shapes can be before communication is required during reconfiguration.

In Section 3 we describe the system assumptions and the problem definition. Section 4 presents and analyzes a distributed algorithm for a chain-to-chain reconfiguration for the case where I and G lie on the same straight line in the plane and intersect in one cell. An extension to generalize the collinear algorithm when I and G are straight chains that occur in any initial relative intersecting orientation is also presented in Section 4. In Section 5 we present lower bounds on the number of moves and time required for the reconfiguration algorithms. We introduce the next step in our solution to the general reconfiguration problem in Section 6, where we describe an algorithm to reconfigure straight chains to a well-defined class of goal configurations. Section 7 provides a discussion of our results and future work.

3. SYSTEM MODEL

3.1 Coordinate system

The plane is partitioned into equal-sized hexagonal cells and labeled using the coordinate system shown in Figure 1.

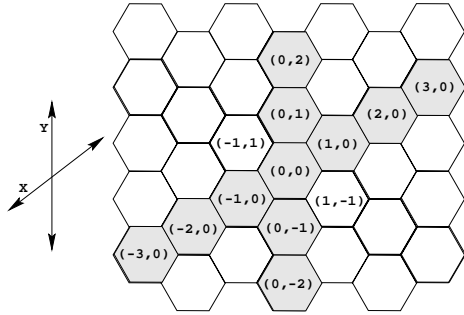


Figure 1: Coordinates in a system of hexagonal cells.

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 .

3.2 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.

- Modules move in lockstep rounds.
- In a round, a module M is capable of moving to an adjacent cell, C_1 , iff
 - cell C_1 is currently empty,
 - module M has a neighbor S that does not move in the round¹ (called the *substrate*) and S is also adjacent to cell C_1 , and
 - the neighboring cell to M on the other side of C_1 from S , C_2 , is empty. See Figure 2 for an example.
- Only one module tries to move into a particular cell in each round.²

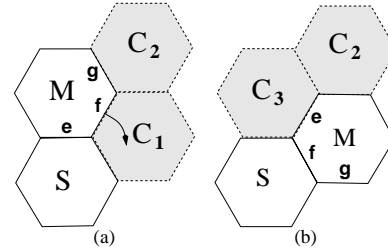


Figure 2: Before (a) and after (b) module movement: M is moving, S is substrate, C_1 , C_2 , and C_3 are empty cells.

3.3 Problem definition

We want a distributed algorithm (local program for each module) that will cause the modules to move from an initial configuration, I , in the plane to a known goal configuration, G .

4. CHAIN ALGORITHMS

In this section, we develop a distributed reconfiguration algorithm for a particular configuration of the system described in Section 3. We focus on reconfiguring a straight-line chain of modules in I to an intersecting straight-line chain of modules in G , where I overlaps G in exactly one cell. Section 4.1 describes a simple algorithm for the case in which I and G are collinear. Section 4.2 analyzes this algorithm. Section 4.3 extends the ideas in the first algorithm for the non-collinear cases.

¹If the algorithm does not ensure that each moving module has an immobile substrate, then the results of the round are unpredictable.

²If the algorithm does not ensure that there are no collisions, then the results of the round are unpredictable.

4.1 Collinear case (case 0)

We classify modules according to their possible connections during an execution of our algorithm in a new classification based on edge contacts. This classification is similar to the one used in [6] that is based on vertex contacts. In Figure 3, modules are classified into three categories (*trapped*, *free*, and *other*) based on the number and orientation of their *contact* and *non-contact* edges. *Non-contact* edges are those on which the module is adjacent to an empty cell and *contact* edges signify that the adjoining cell is occupied by another module. This classification applies to any rotation of a module. Modules in the *trapped* category do not have sufficient adjacent non-contact edges to satisfy hardware constraints on movement (see Section 3.2). Modules classified as *free* are required to move in our algorithm. The *other* category includes modules whose movements are restricted by our algorithm even though their movement would not violate hardware constraints.

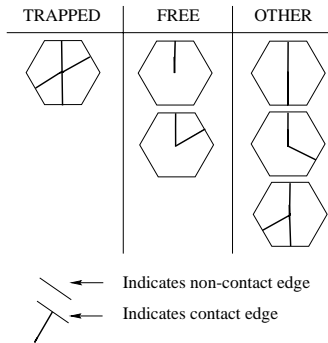


Figure 3: Configuration types possible in case 0 algorithm.

4.1.1 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 and G are collinear and overlap in exactly one cell.

4.1.2 Overview of algorithm

The algorithm works in synchronous rounds. Initially, each module calculates whether it will be moving clockwise (CW) or counter clockwise (CCW) throughout the execution. The modules alternate direction. This is achieved by using the parity of the module's distance from the overlap cell to decide direction of movement. In each round, each module calculates whether it is free (cf. Figure 3) and moves if it is free in the direction calculated initially.

4.1.3 Data structures at each module

- *contacts*: Boolean array indicating which edges have neighboring modules. Assumed to be automatically updated at each round by some lower layer.
- *myCoord*: The coordinates of the module in the plane.
- *goalCell*: Array of all coordinates of cells $\in G$ listed in decreasing order of y coordinate.

- *d*: Variable containing the direction of movement, CW or CCW.
- *flips*: Counter used to determine whether the module is free.

4.1.4 Case 0 reconfiguration algorithm

Code for each module $\notin G$:

```
Initially:
1. if (( $n - LD(myCoord, goalCell[1])$ ) is even)
2.    $d = CCW$ 
3. else  $d = CW$ 

In round  $r = 1, 2, \dots$  :
4. if ( $isFree()$ )
5.   move  $d$ 
```

Procedure $isFree()$:

```
6.  $flips = 0$ 
7. for ( $i = 0$  to 5) do
8.   if ( $contacts[i] \neq contacts[(i + 1) \% 6]$ )
9.      $flips++$ 
10. return ( $flips == 2$ )
```

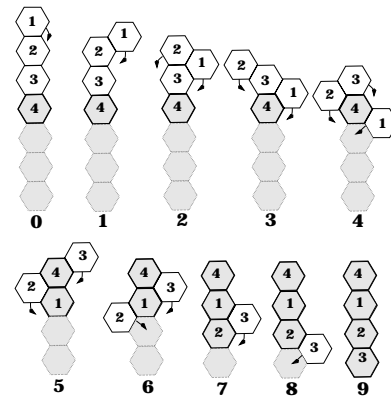


Figure 4: Example reconfiguration.

4.1.5 Example of algorithm operation

In Figure 4 we depict an execution of the case 0 reconfiguration algorithm when $n = 4$. In this figure, occupied cells have solid borders and goal cells are shaded. For purposes of analysis, modules are labeled 1 through 4. Nine rounds are required for this reconfiguration.

4.2 Analysis of reconfiguration algorithm

Without loss of generality, assume that I and G run north-south and I is north of G . Number the cells in I and G from 1 through $2n - 1$ from north to south. We will refer to the module originally in cell i as module i , $1 \leq i \leq n$. We will refer to a cell's neighboring cells as north (N), northeast (NE), southeast (SE), south (S), southwest (SW), and northwest (NW).

THEOREM 4.1. *The case 0 reconfiguration algorithm is correct.*

PROOF. We show that the following properties *I1*–*I3* are invariant throughout the execution. For each i , $1 \leq i \leq n$, and each round $S \geq 1$, at the end of round S ,

- I1: if $S < 2i - 1$, then module i is in cell i ,
- I2: if $S = 2i - 1 + j$, $0 \leq j \leq n - 1$,
 - a) if i is odd, then module i is SE of cell $i + j$,
 - b) if i is even, then module i is SW of cell $i + j$, and
- I3: if $S \geq 2i - 1 + n$, then module i is in cell $i + n$.

We proceed by induction on the number of rounds, S , in the execution. The basis is $S = 0$ (i.e., just before round 1.) In the initial state, *I2* and *I3* are not applicable and *I1* is true by assumption.

For the inductive hypothesis, assume that the invariants hold for round $S - 1$, $S > 0$. Figure 5 illustrates the configuration at the end of round $2k - 1$. In this figure, occupied cells have solid borders and goal cells are shaded.

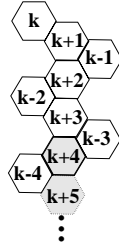


Figure 5: Configuration at end of round $2k - 1$, k is even.

Choose i to be even (without loss of generality.)

Case 1: $S < 2i - 1$.

Thus, $S - 1 < 2i - 2$.

By *I1*, module $i - 1$ is in cell $i - 1$ and module $i + 1$ is in cell $i + 1$ at the end of round $S - 1$. Therefore, module i does not move in round S because it will have contacts on sides N (with module $i - 1$) and S (with module $i + 1$), and possibly on sides NW and SE or sides NE and SW, due to the staggered spacing of the modules in the outer columns. These contacts will be non-contiguous, so module i will not be free in round S . Referring to Figure 5, module $k + 2$ has contacts that correspond to those described for module i .

By *I1*, module i is in cell i at the end of round $S - 1$, so it is still in cell i at the end of round S .

Case 2: $S = 2i - 1 + j$ for some j , $0 \leq j \leq n - 1$.

Thus, $S - 1 = 2i - 1 + (j - 1) = 2(i - 1) + j + 1$.

$j = 0$: Then *I1* implies module i is in cell i at the end of round $S - 1$ and *I2* implies module $i - 1$ is SE of cell i at the end of round $S - 1$, since $S - 1 = 2(i - 1) + 0 + 1$. Then module i is free at the end of round $S - 1$ because

it has contacts only on its S and SE sides. So module i moves CCW, by the code, to be SW of cell i in round S . In Figure 5, module k corresponds to the position described for module i and module $k - 1$ corresponds to the position described for module $i - 1$ in round S .

$j > 0$: Then *I1* implies that module i is in the cell SW of cell $i + (j - 1)$ at the end of $S - 1$. Module i will be free at the end of round $S - 1$ because i will have contacts only on its NE and SE edges, due to the spacing of the cells in the outer columns. Therefore, module i will move CCW in round S to be SW of cell j . Referring to Figure 5, module $k - 2$ is in a position like that described for module i at the end of round $S - 1$.

Case 3: $S \geq 2i - 1 + n$.

Thus, if $S = 2i - 1 + n$, then $S - 1 = 2i - 1 + n - 1 = 2i - 2 + n$. By *I3*, module $i - 1$ is in cell $i + n - 1$ in round $S - 1$, so module $i - 1$ will not move in round $S - 1$ or any round after that, by the code. By *I2*, module i is in the cell SW of cell $i + n - 1$ at the end of round $S - 1$. Therefore, module i is free at the end of round $S - 1$ because it has only one contact, on its NE side, with module $i - 1$. Module i has no other neighbors in round $S - 1$ due to the staggered state of the modules in the outer columns and due to *I3*, which says that if module $i - 1$ is in cell $i + n - 1$, then modules $i - 2, i - 3, \dots, 1$ must have ceased movement before round $S - 1$. So in round S , module i moves into cell $i + n$ in a CCW direction and stops moving in this round by the code. In Figure 5, module $k - 4$ is in a position like that described for module i , and module $k + 4$ is in a position like that of module $i - 1$ at the end of round $S - 1$.

If $S > 2i - 1 + n$, then, since at the end of round S , module i is already in cell $i + n$ by *I3*, then it is still in cell $i + n$ in every round after S . By the code, once a module is in a goal position, it does not move.

These invariants imply that the modules only use three columns. Initially, modules are all in the center column and, during the execution, modules in outer columns are spaced or staggered such that there is an empty cell between any two modules. Invariant *I3* implies that, after round $3(n - 1)$, all modules are in goal positions. \square

THEOREM 4.2. *The case 0 reconfiguration algorithm takes $3(n - 1)$ rounds and makes $n^2 - 1$ module movements.*

PROOF. Invariants *I1* through *I3*, from the proof of Theorem 4.1, imply that the reconfiguration takes $3(n - 1)$ rounds. Invariants *I2* and *I3* show that $n - 1$ of the modules originally in I make $n + 1$ moves each, resulting in $n^2 - 1$ module movements for the whole reconfiguration. \square

The case 0 reconfiguration algorithm works when there is a variable sized overlap in the modules of I and positions in G . With an overlap of size h , the algorithm takes $(n - h)(n + 1)$ moves and $2(n - h) + n - 1$ rounds.

4.3 Non-collinear chain cases

We have developed an extension to the case 0 reconfiguration algorithm to allow the reconfiguration of an initial chain to an overlapping final chain that may intersect the initial chain in any orientation. This extension also does not require any communication between modules, but uses counting techniques combined with knowledge of goal positions to determine the final position of each module.

4.3.1 Algorithm assumptions

For our presentation of the non-collinear algorithm, we assume:

1. I is oriented in a north-south fashion,
2. G is oriented in a southwest-northeast fashion,
3. the northmost module in I is not in G ,
4. if there are cells in I on both the north and south sides of the intersection of I and G , then the longer segment of I is on the north side of the intersection,
5. if there are cells in G on both the east and west sides of I , then the longer goal segment is on the east side of I , and
6. as in Section 4.1.1, initially, each module knows the total number of modules in the system and the coordinates of the goal cells and one module is in each cell of I .

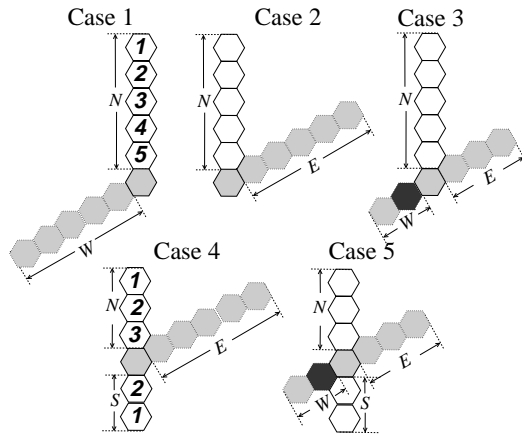


Figure 6: Cases for calculating rotation direction and delay.

The cases for general chain-to-chain reconfiguration are shown in Figure 6, where modules in I have solid borders and goal cells are shaded. Note that if a particular orientation of I and G does not satisfy assumptions 1–5, the coordinates can be translated to form a mirror image or inverted mirror image so that the assumptions are satisfied. For example, in Figure 6, the variant of case 4 with G SW instead of NE of I can be obtained from case 4 by a horizontal and a vertical rotation.

In Figure 6, modules in I in cases 1 through 3 are numbered as shown in case 1 and modules in I in cases 4 and 5 are

numbered as shown in case 4. Throughout the remainder of this paper, we will refer to these numbers as module *positions* in I . N , S , E , and W refer to the number of cells in I or G to the north, south, east, and west of the intersection of I and G . The darker shaded goal cells in cases 3 and 5 are used by certain modules to the north to “cut through” the west side of G to fill in goal cells on the east side of G , as discussed below.

The number of possible configuration types is larger in the general case, as shown in Figure 7. 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.

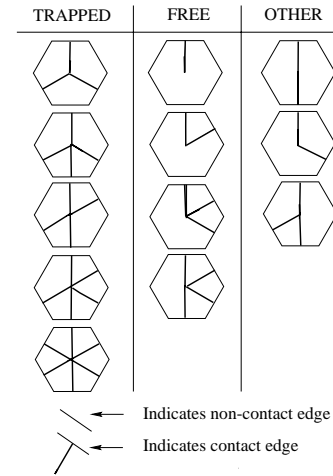


Figure 7: Configuration types possible in general chain cases.

The algorithms for cases 1 through 5 are similar to that for case 0. The differences are

- determining whether a module is free,
- calculating the *direction* in which to move, and
- calculating the *delay*, i.e., how long to wait after becoming free until beginning to move.

The *isFree()* procedure presented in Section 4.1 must be modified to take into account the increased number of configuration types available (cf. Figure 7). This is accomplished at the module in position i in I by counting its contact edges and returning true if

- the neighboring cell in position $i - 1$ is unoccupied, $flips = 2$, and the number of contact edges < 5 ,

and false otherwise.

Modules calculate direction and delay by counting modules with lower initial positions as they pass. The progress made on filling in goal cells in the east and west sections of G is determined locally by maintaining separate tallies ($numE$ and $numW$) for modules passing on the east and west side.

The algorithm schema is:

Code for module $\notin G$:

Initially, $numW = numE = 0$.

In round $r = 1, 2, \dots$:

If module is north of G :

- if N neighboring cell is occupied or newly vacated, then
 - if NW neighboring cell is newly occupied, then $numW++$
 - if NE neighboring cell is newly occupied, then $numE++$
- if N neighboring cell is newly vacated, then
 - calculate *direction* and *delay* using $numW$ and $numE$

If module is south of G :

- if S neighboring cell is occupied or newly vacated, then
 - if SW neighboring cell is newly occupied, then $numW++$
 - if SE neighboring cell is newly occupied, then $numE++$
- if S neighboring cell is newly vacated, then
 - calculate *direction* and *delay* using $numW$ and $numE$

If *isFree()*:

- if $delay = 0$ then move *direction*
- else $delay--$

The goals for the choice of *direction* and *delay* at a module are to

1. avoid collisions,
2. avoid deadlock (a situation where the module is not free but not in a goal position), and
3. try to minimize the total elapsed time (i.e., maximize parallelism.)

From Figure 6, we can see that, in all cases shown, modules originally in I will be rotating toward either an obtuse or an acute angle intersection with G . In case 0, modules will begin rotating in the CW direction. Define the following patterns of direction and delay:

- *1-directional obtuse*: Modules in I move in same direction, toward the obtuse angle intersection with G , with no delay for the module in position 1 and a delay of 1 for all other modules.
- *1-directional acute*³: Module 1 in I starts moving toward the acute angle intersection with G with no delay. Subsequent modules move in the same direction with a delay of 2.

³This pattern is not used until Section 6.

- *2-directional obtuse*: Module 1 in I starts moving toward the obtuse angle intersection with G with no delay. Subsequent modules alternate direction with no delay.
- *2-directional acute*: Module 1 in I starts moving toward the acute angle intersection with G with no delay. Subsequent modules alternate direction with a delay of 1.

Since modules can only move into cells that are empty, there must be at least one empty cell between all modules moving in the same direction over modules in I toward G . For modules rotating toward the acute angle intersection, there must be at least 2 empty cells between all modules moving in the same direction over modules in I toward G to avoid deadlock in the corner.

Case 0: *2-directional obtuse*, starting in CW direction.

Case 1: *2-directional obtuse*.

Case 2: *2-directional acute*.

Case 3: If module is north of G and has position $\leq E$, pattern = *2-directional acute*. Modules north of G in positions $\leq E$ and moving in CCW direction move through goal cell with darker shading in Figure 6. If module is north of G and has position $> E$, pattern = *1-directional obtuse*.

Case 4: If module is south of G , pattern = *1-directional obtuse*. If module is north of G and position is 1, delay = $(3 * S) - N + 1$ and rotation is toward the acute angle intersection. If module is north of G and position > 1 , pattern = *2-directional acute*.

Case 5: If module is south of G , pattern = *1-directional obtuse*. If module is north of G and $N = W$, then pattern = *1-directional obtuse*. Otherwise, if module is north of G and $N > W$:

- If position is 1, delay = $(3 * S) - N + 1$ and rotation is toward the acute angle intersection.
- If $1 < \text{position} \leq (E - S)$, pattern = *2-directional acute*. Modules on the north of G in positions $\leq (E - S)$ and moving in CCW direction toward the obtuse angle intersection move through the darker shaded goal cell in Figure 6.
- If position $> (E - S)$, pattern = *1-directional obtuse*.

The number of moves made and number of rounds used in case 0 was discussed in Section 4.2. It is easy to show that case 1 uses the same number of rounds and moves. Like cases 0 and 1, cases 2 through 5 use $\Theta(n^2)$ movements and $\Theta(n)$ rounds, although the constants will vary based on the relative values of N , S , E , and W .

Simple extensions to the algorithm allow the reconfiguration of a chain in any initial location to any final location by calculating intermediate configurations that lead to eventual intersection and then running the algorithm described in this section.

5. LOWER BOUND PROOFS

The problem of finding bounds for the number of moves needed to reconfigure a metamorphic system was addressed for general configurations in [3]. We show a tight lower bound on the number of moves required for the reconfiguration of cases 0 and 1 and give proof sketches that show cases 2 through 5 are asymptotically tight. We also consider the lower bound for the total elapsed time of this reconfiguration, a measure that was not addressed in [3]. Our bounds are asymptotically tight in this measure.

5.1 Number of module movements

THEOREM 5.1. *Any algorithm to reconfigure a metamorphic system in cases 0 or 1 under the system assumptions given must cause at least $n^2 - 1$ module movements.*

PROOF. Number the cells occupied by the modules in I and the goal positions in G as in Section 4.2, from 1 to $2n-1$.

Suppose the module initially in cell i ends up in goal cell p_i , $1 \leq i \leq n$. Note that each p_i is between n and $2n-1$ and they are all unique. Also note that the distance between i and p_i is $p_i - i$.

By the constraints on module movement, we claim that the minimum number of moves required for a module to move from cell i to cell p_i is $p_i - i + 1$, if $i \neq p_i$, and 0 if $i = p_i$. Note that only one value of i can have $p_i = i$, namely $i = n$.

To see why the minimum number of moves is $p_i - i + 1$ when $i \neq p_i$, observe that, if a module in cell i is to traverse a shortest path to cell p_i , it needs a stationary substrate module next to it initially so it can roll into the empty cell that lies one cell closer to cell p_i . Therefore, some module has to move into that supporting position.

So the total number of module movements is at least:

$$\begin{aligned} \sum_{i=1}^{n-1} (p_i - i + 1) &= \sum_{i=1}^{n-1} p_i - \sum_{i=1}^{n-1} i + n - 1 \\ &= \sum_{i=n+1}^{2n-1} i - \sum_{i=1}^{n-1} i + n - 1 = n^2 - 1 \end{aligned}$$

□

Since in case 2 module i must move at least $n - i$ cells to reach any goal position, we have the following theorem.

THEOREM 5.2. *Any algorithm to reconfigure a metamorphic system as in case 2 under the system assumptions given must cause at least $\frac{n^2-n}{2}$ module movements.*

Referring to Figure 6, for case 3, Theorems 5.1 and 5.2 say the number of movements is at least $(W^2 - 1) + \frac{E^2 - E}{2}$. For case 4, we have that the number of movements is at least

$(S^2 - 1) + \frac{N^2 - N}{2}$. For case 5, the number of movements is at least $\frac{N^2 - N}{2} + \frac{S^2 - S}{2}$. Therefore, we have:

THEOREM 5.3. *Any algorithm to reconfigure a metamorphic system as in cases 3, 4, and 5 under the system assumptions given must cause $\Omega(n^2)$ module movements.*

5.2 Number of rounds

We now show lower bounds on the number of rounds required for reconfiguration. Each module has a final position. Given a state of the system, let v_i be the lattice distance between module i 's current and final positions, for $1 \leq i \leq n$.

THEOREM 5.4. *Any algorithm to reconfigure a metamorphic system from a chain to an intersecting chain under the system assumptions given must take at least $\frac{D}{\lfloor \frac{n}{2} \rfloor}$ rounds, where D is the sum of all the v_i s in the starting state.*

PROOF. Clearly, at the end of the reconfiguration, the sum of all the v_i s is 0. This is because, at the end of the reconfiguration, each module in I will be in its final position, and therefore, each $v_i = 0$. Note that each v_i can vary by at most one in a round. Then we make the following claim:

CLAIM 1. *The maximum number of processors that can decrease their v_i in one round is $\lfloor \frac{n}{2} \rfloor$.*

PROOF. The maximum number of modules that can move in one round is $\lfloor \frac{3}{4}n \rfloor$. However, at most two out of every four modules can decrease their v_i during that round, as can be shown by an exhaustive listing of all possible moves of three modules over one substrate. □

The lower bound on the number of moves needed to reconfigure is D . So the number of rounds needed is at least $\frac{D}{\lfloor \frac{n}{2} \rfloor}$. □

To apply Theorem 5.4 to the cases in Figure 6, we must calculate D . An argument similar to that in the proof of Theorem 5.1 shows that D is $n^2 - n$ for cases 0 and 1. The arguments for Theorems 5.2 and 5.3 show that D is $\frac{n^2-n}{2}$ for case 2 and $\Theta(n^2)$ for cases 3 through 5.

Thus, we have:

COROLLARY 5.5. *The minimum number of rounds is*

- $2(n-1)$ for cases 0 and 1,
- $n-1$ for case 2, and
- $\Omega(n)$ for cases 3 through 5.

6. MORE GENERAL RECONFIGURATION

In this section we present a first step in extending our reconfiguration of chains to more general shapes of G . We define an admissible class of configurations for G and give a distributed algorithm to accomplish the reconfiguration. Since we assume that modules have no initial global knowledge of I , our admissible class of configurations for G is more restrictive than that presented in [8].

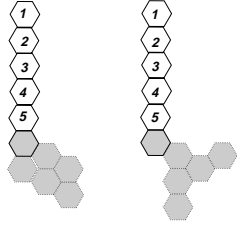


Figure 8: Example relationships of I and G .

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 southmost module of I and nowhere else, as shown in Figure 8 (if I and G do not satisfy these conditions, then I can be moved using the method sketched at the end of Section 4). In all figures in this section, modules in I have solid boundaries and cells in G are shaded.

Let G_1, G_2, \dots, 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, \dots, 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 eastmost column of G (G_m).

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 G_i , $1 \leq i \leq m$, is contiguous and
2. there exists an admissible substrate path in G .

Intuitively, an admissible substrate path is a chain of goal cells whose surface allows the movement of modules without

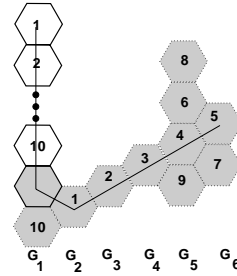


Figure 9: Example reconfiguration (G is admissible).

collision or deadlock, provided the choices of rotation and delay are correct.

Figure 9 depicts an example of an admissible configuration of G , where the line through I and G is an admissible substrate path. Figure 10 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.

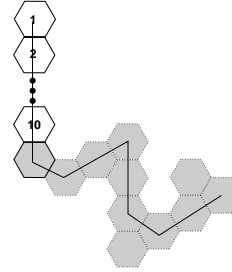


Figure 10: Example of inadmissible G .

We assume that the modules in I know the coordinates of an admissible substrate path p in G . In the full paper we will present efficient algorithms to find admissible substrate paths in G that evenly partition the goal cells into those above and those below the substrate path. Even partitioning of the goal cells is desirable since it allows modules to alternate directions, maximizing parallelism.

Each module in I calculates the *designated column* G_i in which it will stop in a goal position. 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 9 has numbered goal cells depicting how initial module positions correspond to final goal positions.

The reconfiguration proceeds as follows:

- For modules in positions 1 through $|p|$:
 - Modules use *1-directional acute* pattern (see Section 4.3) in CW direction.
 - When a module is in a goal cell that is in its designated column on the substrate path, it stops in that cell.
- For modules in positions $> |p|$:

- Modules use *2-directional acute* pattern until all cells on one side of p are filled. After this, modules use a *1-directional acute* pattern (see Section 4.3), 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 a goal cell in its designated column, it stops.
- Once a module stops in a goal cell for a round it never moves out of that goal cell.

7. 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. 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 collision and deadlock on the fly, which we are currently testing and refining.

The reconfiguration algorithms described in this paper are simplistic, applying only to a narrow range of reconfiguration options. On the other hand, the algorithms are completely distributed, requiring no communication between modules. The algorithms were shown to be optimal or asymptotically optimal in terms of number of movements and asymptotically optimal in the reconfiguration time used. We conjecture that the collinear algorithm is optimal in the time used, but have not yet been able to improve the lower bound.

We are working on optimizations to the algorithm framework outlined in Section 6. One of our future goals is to develop algorithms to identify admissible configurations of G and divide non-admissible configurations into admissible subconfigurations.

For the more general case of reconfiguration, the modules may need to communicate with each other (if all other hardware constraints remain as described). Modules may also need a more global picture of the initial and intermediate configurations to make local planning strategies possible. We believe that the chain reconfiguration algorithms presented in this paper will provide a building block for distributed reconfiguration of more arbitrary initial and final configurations.

The goal of our future work is to develop more complex distributed reconfiguration strategies from building blocks like the one presented in this paper. During this development process, we hope to further refine our system model by discovering which assumptions are sufficient and necessary to reconfigure such a system.

8. 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] M. Yim. A reconfigurable modular robot with many modes of locomotion. In *Proc. of Intl. Conf. on Advanced Mechatronics*, pages 283–288, 1993.
- [12] M. Yim, J. Lamping, E. Mao, and J. G. Chase. Rhombic dodecahedron shape for self-assembling robots. SPL TechReport P9710777, Xerox PARC, 1997.
- [13] 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.