Distrib. Comput. (2004) 17: 171–189
Digital Object Identifier (DOI) 10.1007/s00446-003-0103-y

DISTRIBUTED COMPUTING

Distributed reconfiguration of metamorphic robot chains [⋆]

Jennifer E. Walter[1], **Jennifer L. Welch**[2], **Nancy M. Amato**[2]

[1] Department of Computer Science, Vassar College, Poughkeepsie, NY 12604-0351, USA (e-mail: walter@cs.vassar.edu)
[2] Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA (e-mail: {welch,amato}@cs.tamu.edu)

Received: October 28, 2002 / Accepted: October 31, 2003
Published online: March 1, 2004 – © Springer-Verlag 2004

Abstract. The problem we address is the distributed reconfiguration of a planar metamorphic robotic system composed of any number of hexagonal modules. After presenting a framework for classifying motion planning algorithms for metamorphic robotic systems, we describe distributed algorithms for reconfiguring a straight chain of hexagonal modules to any intersecting straight chain configuration. We prove our algorithms are correct, and show that they are either optimal or asymptotically optimal in the number of moves and asymptotically optimal in the time required for parallel reconfiguration.

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. Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as bridge building, satellite recovery, or tumor excision [24]. This paper builds on the work of several researchers in the field of robotics, including [7,12,15,17,24,37]. A robotic system that changes its shape due to individual module motion has been called *self-reconfigurable* [15] or *metamorphic* [7].

A self-reconfigurable robotic system is a collection of independently controlled, mobile modules, each of which has the ability to adhere to, disconnect from, and move around adjacent modules. Metamorphic robotic systems, a subset of self-reconfigurable systems, possess the following additional properties [6]:

- Each module is identical in structure, motion constraints, and computing capabilities.
- Modules are space filling polygons or polyhedra [20] that can be packed densely, i.e., packed so that gaps between modules are as small as possible, allowing these systems to form two and three dimensional solids [1].

In the systems we consider, modules achieve locomotion by moving over a substrate composed of other modules. The mechanics of locomotion depend on the hardware and can include module deformation to crawl over neighboring modules [10, 24] or to expand and contract to slide over neighbors [28]. Alternatively, moving modules may be constrained to rigidly maintain their original shape, requiring them to roll over or lift themselves around neighboring modules or other surfaces [14,17–19,29,32,39,42].

Metamorphic robotic systems are being widely studied because their shape changing abilities make them potentially useful for a larger set of tasks than conventional robotic systems. Since metamorphic robotic systems are intended to be composed of a large number of duplicate modules, they can potentially tolerate multiple processor failures and may be useful in environments that are not amenable to direct human observation and control (e.g., interplanetary space, undersea depths). Such systems can also be *self-repairing* [20], allowing damaged modules to be replaced by identical working counterparts.

The motion planning problem for a metamorphic robotic system is to determine a sequence of module motions required to change a given initial configuration (I) to a desired goal configuration (G). In this paper, we present algorithms and lower bound proofs for particular cases of this reconfiguration: collinear and non-collinear chain-to-chain metamorphosis. Figure 1 illustrates the reconfiguration cases presented.

Developers of existing self-reconfigurable robotic systems, e.g., [5,7,9,12,13,15,17,24,28,30,31,37] have devised motion planning strategies specific to the hardware constraints

[⋆] A preliminary version of this paper appeared in the *Proc. of the 19th ACM Symposium on Principles of Distributed Computing*, July 2000, pages 171–180.
Correspondence to: J. Walter
The work of N. Amato and J. Walter was supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACR-0081510, ACR-0113971, CCR-0113974, EIA-9810937, EIA-0079874, by the Texas Higher Education Coordinating Board grant ARP-036327-017, and by the DOE ASCI ASAP program grant B347886. The work of J. Walter was supported in part by Department of Education GAANN and GE Faculty of the Future fellowships.

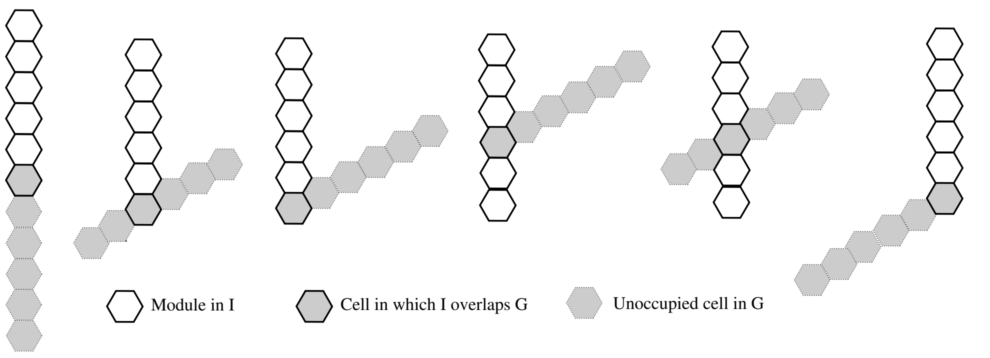


Fig. 1. Robot chain reconfiguration scenarios considered in this paper

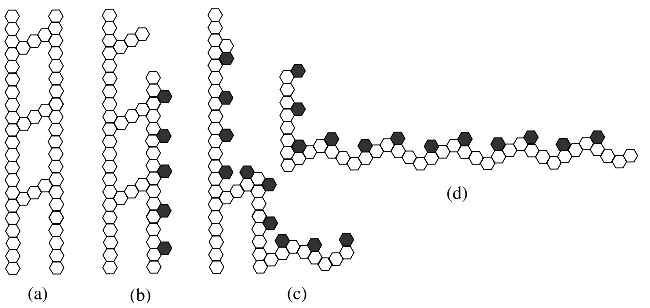


Fig. 2a–d. Application of straight chain reconfiguration algorithms – scaffold to bridge structure. Execution progresses in time from (a) to (d). Stationary modules are white and moving modules are black

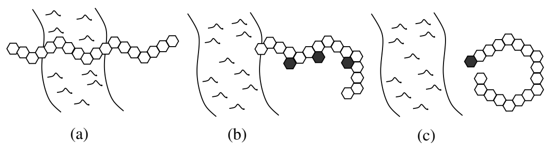


Fig. 3a–c. Application of straight chain reconfiguration algorithms – bridge to corral structure. Execution progresses in time from (a) to (c). Stationary modules are white and moving modules are black

of their prototype robots. Many existing motion planning strategies rely on centralized motion planning algorithms to plan and supervise the motion of the system components [4, 7,15,21,24,28,37]. Others, such as [16,17,19,29,33,39–42], use distributed approaches that rely on heuristic approximations and/or require message passing between modules to coordinate module actions in each step of the reconfiguration process.

The chain-to-chain reconfiguration algorithms presented in this paper are *substrate* [4], or *space filling* [20] reconfigurations, where individual modules move between points on a lattice composed of identical modules. Our approach should not be confused with the *closed-chain* [4], or *linear string* [20] approaches used in [4,12,13,16,20,22,29,38], where a serial chain of heterogenous modules forms a multijoint robot.

We consider a deterministic distributed motion planning strategy for the configurations shown in Fig. 1, given the assumption of initial global knowledge of G. We focus on a system composed of planar, hexagonal robotic modules as described in [7]. Our distributed approach offers the benefits of localized decision making and the potential for greater system fault tolerance. Additionally, our algorithms require no intermodule message passing during reconfiguration and ensure collision- and deadlock-free execution.

The algorithms described in this paper are intended as building blocks, to be used for the reconfiguration of the system into arbitrary shapes. Figure 2 depicts an example of a scaffold to bridge reconfiguration, realized by sequences of straight chain reconfigurations. Figure 3 is an aerial view of a bridge to corral reconfiguration that uses a series of straight chain reconfigurations. In these examples, the modules would need to have global knowledge of the initial and goal configurations.

In our recent work [35,36], we have extended these chain reconfiguration algorithms to allow the transformation of a straight chain of modules to an *admissible* goal configuration. Informally, an admissible goal configuration has arbitrary shape with the restriction that the goal columns are composed of contiguous cells in some orientation and that the goal contains a bisecting chain (called an *admissible substrate path*) of cells that can be filled in and traversed by modules without collision or deadlock. The reconfiguration of a straight chain I to an admissible substrate path is a direct application of the chain-to-chain algorithm presented in this paper. A more formal definition of an admissible goal configuration can be found in [35] and is beyond the scope of this paper.

1.1 Problem definition

Reconfiguration algorithms for metamorphic robotic systems cause the modules to move from an initial configuration, I, in the plane to a goal configuration, G. In this section we classify these algorithms according to particular properties they possess and by constraints they operate under.

We classify reconfiguration algorithms according to the following kinds of properties:

- *Hardware properties:* module shape and motion constraints.
- *Configuration properties:* feasible shapes, connectivity, and other requirements on I and G, presence of obstacles in the reconfiguration space.
- *Communication and control properties:* centralized versus distributed, synchronous versus asynchronous, communication versus contact-only information.
- *Algorithmic properties:* deterministic versus probabilistic, global connectivity requirements, stopping conditions.

Hardware properties: Reconfiguration algorithms have been developed for metamorphic robotic modules of various shapes, including hexagons, squares, cubes, and rhombic dodecahedra[1]. The algorithms are particular to the motion constraints of the modules. In some cases, the modules are deformable, compressing or elongating to facilitate movement and then returning to their original resting shape. In other cases, the modules are rigid and do not change shape during movement. In general, deformable modules have less restrictive motion constraints than their rigid counterparts because deformable modules are able to squeeze through smaller gaps in the lattice.

Reconfiguration algorithms differ in the assumptions they make on the capability of the component modules. Modules may be "individually mobile" (IM), i.e., have the capacity to move only themselves over a lattice of other, identical modules. In other systems, an individual module is capable of pushing or carrying a subset of modules besides itself, in which case we say modules are "other movable" (OM). The former set is a proper subset of the latter (IM $\subset$ OM). The advantage of modules being OM is that many modules can move in a single time step under the power of one module. The disadvantage of OM systems is that each module needs additional power to lift, carry, or push adjacent modules. The effects of friction-generated heat on module contact surfaces is also a concern in systems with OM modules.

Definitions of feasible traversal surfaces also vary. Modules may be limited to moving over other modules or they may be able to move over arbitrary surfaces and obstacles. Movement of modules may be possible only on the perimeter of a configuration or modules may be designed to "tunnel through" a mass composed of other identical modules.

Configuration properties: The reconfiguration space is generally divided into discrete units that are identical to the size and shape of the modules.

Reconfiguration algorithms often place restrictions on the relative positions of I and G, e.g., I and G may be required to overlap or have particular shapes and orientations. When I and G overlap, some of the modules in G may be treated as a "fixed base", meaning that none of these modules move during the reconfiguration. In general, restrictions on the relative positions of I and G limit the number of situations in which the algorithm is useful.

Requirements may be placed on feasible shapes of I and G for successful reconfigurations, e.g., "holes" or narrow tunnels in I and G may be disallowed. Most approaches require an exact description of the shape and position of G. However, other reconfiguration approaches do not define an exact position, requiring only the final shape of G to match a-priori specifications or specifying only a particular goal cell or cells that must be reached. Others do not define an exact shape for G and aim instead at creating a structure with the correct morphology required to achieve some task.

Ideally, a reconfiguration algorithm should place no restrictions either on the shapes or relative positions of I and G. However, the problem of finding the optimal sequence of moves from I to G, both of arbitrary shape, has been shown to take time that is exponential in the number of modules [8]. Therefore, either heuristics are used to find a near-optimal probabilistic solution, or the shapes of I and G are restricted to allow a successful deterministic approach. The former strategy suffers from the possibility that the reconfiguration will not be successful, and the latter from lack of generality and extra time spent in pre-processing.

Since modules in a metamorphic system are homogeneous, it is generally not the case that modules need to fill particular "labeled" positions in G, except possibly for cells that overlap I and G. Systems that do impose requirements on individual module positions in G have more limited possibilities for module movement during reconfiguration and are generally composed of heterogenous module types.

The presence of obstacles in the reconfiguration space is another variable property considered in some reconfiguration strategies.

Communication and control properties: In *centralized* reconfiguration strategies, movements of individual modules are calculated using global knowledge of the system state throughout reconfiguration. Centralized strategies suffer from the existence of a single point of failure as well as from the necessity of maintaining a global view of the system.

Distributed reconfiguration algorithms allow modules to use only local information and possibly message passing to determine moves that will accomplish global reconfiguration. Distributed algorithms allow as many modules as possible to move in parallel and can provide greater fault tolerance due to the lack of a single point of failure. On the other hand, these algorithms may fail to prevent collision and deadlock and may therefore not accomplish the complete reconfiguration. As in any distributed system, modules may or may not have unique identifiers. Also, these strategies vary according to the initial knowledge each processor has of the system. For example, prior to execution, each module may know any or all of the following:

– the total number of modules,
– the coordinates of all cells in the goal configuration, or
– the coordinates of all cells in the initial configuration.

Algorithms differ in the amount of inter-module communication that is required during reconfiguration. Certain algorithms use no communication because each module contains the full implementation of the reconfiguration process, so the entire state of the system is known by each module at all points in the execution. At the other extreme are systems in which modules need to use extensive message passing in order to carry out the reconfiguration. An intermediate approach, like our own, requires modules to know the initial coordinates of G and the ability to dynamically sense on which sides they contact other modules. In this approach, no message passing is needed, and only local knowledge is necessary at each module during reconfiguration. Inter-module message passing can facilitate reconfiguration with less pre-processing and fewer restrictions on the initial configurations of I and G. Message passing can also provide the coordination necessary for asynchronous module movement without collision. The advantage of restricting inter-module message passing during reconfiguration is that less module energy needs to be spent generating and processing messages and reconfiguration is less subject to outside interference in the form of message blocking. How-

[1] A rhombic dodecahedron is a polygon with twelve identical faces, each of which is a rhombus [2].

ever, more pre-processing is generally necessary when communication is limited.

Distributed reconfiguration algorithms may move modules in rounds, since coordination of movement is necessary to prevent module collision when multiple modules can move concurrently. These rounds of movement may be synchronized according to a global clock, or they may be interspersed with one or more communication phases. Alternatively, movement may be coordinated locally, allowing for asynchronous module movement. Synchronized reconfiguration requires more hardware overhead (e.g., synchronized clocks) and therefore tends to be more difficult to ensure in a distributed system. On the other hand, asynchronous systems tend to need more communication to ensure no collision or deadlock occurs.

Algorithmic properties: Deterministic planners provide motion plans that avoid module collision and can guarantee successful reconfiguration, given that the shape of I and G meet certain pre-defined characteristics. The main disadvantage of deterministic planners is that they generally impose restrictions on the shapes of I and G, thereby limiting the situations in which they are useful.

Probabilistic motion planners generally use a stochastic process to drive the reconfiguration, relying on local message passing to ensure two modules do not attempt to move into the same cell at the same time. Successful completion of general reconfigurations sometimes cannot be guaranteed with probabilistic planners due to blocking problems and module deadlock arising from the random nature of module movement.

Complete planners are both deterministic and are able to reconfigure the system into any arbitrary shape.

Reconfiguration algorithms may also enforce other properties on the execution. We define two of these below:

- In an *always-connected* reconfiguration algorithm, the modules form a single connected component at all times. This property is often motivated by the practical constraint that there must be a fixed power source for the modules. For most metamorphic robotic systems, the motion constraints on the modules require them to move over an immobile substrate composed of other identical modules. In such systems, a subset of the modules must be static during each round.
- A *goal-stopping* reconfiguration algorithm does not allow any module occupying a goal position to move. Because any module in I can fill any goal cell in G, reaching a goal cell is a convenient, locally detectable stopping point for a module executing its local copy of a reconfiguration algorithm.

Both the goal-stopping and the always-connected properties listed above impose restrictions on the algorithms that can result in less efficient reconfiguration, as we show in Sect. 5. However, these properties have the advantage of being enforceable on the local level by individual modules.

The complexity measures of interest for these reconfiguration algorithms include the number of module movements, the number of rounds or overall time used, the number of messages, and the computation time for motion planning.

1.2 Related work

In this section, we briefly survey related work on metamorphic robots. We consider the class of two- and three-dimensional systems known as *space filling* [20] or *substrate* [4] systems, where individual modules can move over a lattice composed of other modules to accomplish reconfiguration. Our discussion is organized by module type and communication and control modes. Table 1 summarizes the distinguishing properties of algorithms that reconfigure systems of metamorphic robots.

Hexagonal metamorphic robots

Upper and lower bounds on reconfiguration, feasibility of reconfiguration:
Chirikjian and Pamecha [8] present upper and lower bounds on the number of moves for general reconfiguration. They show an upper bound on the minimal number of moves that 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. 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.
Dumitrescu et al. [11] present upper and lower bounds on the speed of locomotion for particular formations of hexagonal robots and provide results on the number of rounds needed for any module in one of these formations to reach a particular target cell in a goal configuration.
In [21], Nguyen et al. analyze the number of moves necessary for specific shapes of I and G. They 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 and feasible configurations based on the rigid nature of their modules and use knowledge about the initial configuration to plan the reconfiguration process.
Centralized reconfiguration algorithms:
Chirikjian and Pamecha [10] and Pamecha et al. [24] present algorithms that use the distance between all modules in I and cells in G to drive the reconfiguration. This distance metric is applied to system self-reconfiguration using a simulated annealing technique to drive the process towards completion.
Distributed reconfiguration algorithms:
Murata et al. in [17] present an approach in which each module senses its own connection type and classifies itself by the finite number of modules that it physically contacts. This approach uses random local motions to converge toward the goal configuration, a slow process that appears impractical for large configurations. This scheme ignores the consequences of module collision and does not distinguish the relative location of I and G in the plane, i.e., two configurations are the same if the modules composing them have the same connections.

Square metamorphic robots

Upper and lower bounds on reconfiguration:
Dumitrescu et al. [11] present upper and lower bounds on the speed of locomotion for particular formations of square robots.

Centralized reconfiguration algorithms:
Chiang and Chirikjian [6] present bisecting techniques used to find intermediate configurations between any given initial and final configurations. They combine these techniques with the simulated annealing algorithm of Pamecha et al. [24] to reconfigure the system.
Distributed reconfiguration algorithms:
The rigid square modules developed by Hosokawa et al. [14] move as a result of being lifted, sliding, and pivoting over other rigid square modules by built-in rotatable arms. These algorithms use interactions between neighboring modules to generate specific global formations.

Cubic metamorphic robots

Centralized reconfiguration algorithms:
Rus and Vona [28] present algorithms to plan the reconfiguration of a system of compressible cubic modules that can expand and contract, resulting in the sliding movement of a module over its neighbors. Feasible configurations of I and G are specific to the alignment of the cubic modules organized into meta-modules called "grains". Reconfiguration algorithms are presented for feasible configurations. In these algorithms, internal modules can contract and pull surface modules inward or expand to push surface modules outward.
Distributed reconfiguration algorithms:
A set of distributed motion planning algorithms for compressible cubic modules is presented by Butler et al. in [3]. These algorithms allow modules to move asynchronously after a planning phase that requires extensive inter-module communication. The reconfiguration algorithm also involves message passing to ensure correctness. The definition of feasible configurations of I and G is similar to that given in [28]. In [41], Yoshida et al. present a distributed reconfiguration algorithm for a system of rigid skeletal cubes that move in temporary pairwise collaborations, with one module in a pair rotating the other module in the pair into a new position. This algorithm features a stochastic relaxation process that allows the system to converge toward a particular goal configuration by locally searching for a proper unit motion over many degrees of freedom.

Rhombic dodecahedral metamorphic robots

Distributed reconfiguration algorithms:
Yim et al. [39] present a probabilistic motion planning strategy in which 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 make progress toward the goal configuration. While this algorithm allows parallel motion of modules, it does not consider the consequences of a module moving over another moving module in the same time step, an occurrence which could possibly violate specified motion constraints by causing a partition in the system.
Zhang et al. [42] present several heuristic approximation algorithms which can potentially decrease the distance from the initial to the goal configuration. In this two phase approach, modules use neighbor-to-neighbor communication 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 round of movement.
Bojinov et al. [1,2] describe distributed probabilistic algorithms to control the reconfiguration of systems of rhombic dodecahedra into arbitrary target configurations that have properties required for a specific task.

General self-reconfigurable modules

Many self-reconfigurable robotic systems are composed of modules that do not strictly adhere to the definition of metamorphic robots and are therefore not included in Table 1. For example, the *self-reconfiguring molecule* [15], moves over a cubic lattice, but the basic modules are not regular polyhedra and are therefore not space filling. Also, the individual components of these molecules are not identical, being composed of male and female counterparts.

As mentioned previously, the closed-chain [4] reconfigurable systems *CEBOT* [12], *CONRO* [29], *Tetrobot* [13,16], and *I-Cube* [32] use reconfiguration strategies that apply to strings of modules, not individual modules, and are not readily comparable to our motion planning algorithms.

In *meta-module* systems [21,26,28,33], each unit in the lattice is composed of a number of individual modules, moving in formation or breaking up to accomplish "tunneling" reconfigurations. The reconfiguration tasks in these meta-module systems are simplified [21,33], and therefore the motion planning problem is not as hard as the one for individual modules. Since we are concentrating on classifying algorithms for modules that most closely resemble our own, we therefore do not include meta-module systems in our classification in Table 1. Vassilvitskii et al. [33] present a complete, parallel motion planning algorithm for a system composed of *Telecubes*, metamodules formed from eight cubic metamorphic modules similar in design to those developed by Rus et al. [28]. It is interesting to note that completeness of the reconfiguration space can be guaranteed in these metamodule systems, but not by any of the algorithms listed in Table 1.

1.3 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 consider two dimensional, hexagonal modules like those described by Chirikjian in [7], using their definition of lattice distance between modules in the plane. We refer the reader to [7,25,27,30] for discussions of the systems issues involved with metamorphic modules, since these issues are beyond the scope of this paper.

We believe one contribution of our work is how our system model abstracts from specific hardware details about the modules. Our proposed scheme uses a new classification of module types based on connected edges similar to the classification presented by Murata et al. in [17] 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

Table 1. Summary of existing algorithms for metamorphic system reconfiguration using the categories described in Sect. 1.1. RD = rhombic dodecahedra, IM = independently mobile, OM = module movable by other modules

Algorithm	Hardware	Communication and Control	Configuration	Algorithmic
Chirikjian et al. [10] Pamecha et al. [24]	hexagons, deformable, IM	centralized	I arbitrary, G arbitrary, I and G overlap, G position fixed	probabilistic
Nguyen et al. [21]	hexagons, rigid, IM	centralized	I feasible, G chain, I and G overlap, G position not fixed	deterministic
Murata et al. [17]	hexagons, rigid, IM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position not fixed	probabilistic
Chiang and Chirikjian [6]	squares, rigid, IM	centralized	I arbitrary, G arbitrary, I and G overlap, G position fixed	probabilistic
Hosokawa et al. [14]	squares, rigid, OM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position not fixed	probabilistic
Rus and Vona [28]	cubes, deformable, OM	centralized	I feasible, G feasible, overlap unnecessary, G position fixed	deterministic
Butler et al. [3]	cubes, deformable, OM	distributed, asynchronous, message passing	I feasible, G feasible, overlap unnecessary, G position not fixed	deterministic
Yoshida et al. [41]	cubes, rigid, OM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position not fixed	probabilistic
Bojinov et al. [1,2]	RD, rigid, IM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position not fixed	probabilistic
Yim et al. [39]	RD, rigid, IM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position not fixed	probabilistic
Zhang et al. [42]	RD, rigid IM	distributed, asynchronous, message passing	I arbitrary, G arbitrary, overlap unnecessary, G position fixed	probabilistic
Our approach (this paper)	hexagons, deformable, IM	distributed, synchronous, no message passing	I straight chain, G straight chain, I and G overlap, G position fixed	deterministic
Walter et al. [35]	hexagons, deformable, IM	distributed, synchronous, no message passing	I straight chain, G admissible, I and G overlap, G position fixed	deterministic

plan that avoids module collision. Modules in our system are completely interchangeable and it should be noted that the correctness of our algorithms is not guaranteed if modules must fill particular positions in G. We analyze our algorithms in terms of number of module movements and number of rounds used.

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

- *Our algorithms use minimal communication.* 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. We show that reconfiguration in scenarios like the ones presented in this paper can be accomplished using algorithms that do not require any message passing. Therefore, our algorithms are more communication efficient than the distributed approaches of [3,17,39–42]. One of our future goals is to determine how complex the configuration shapes can be before message passing is required during reconfiguration.
- *Our algorithms are deterministic.* Unlike the probabilistic approaches of [17,39–42], our motion planning strategy will always produce the intended reconfiguration, provided I and G are straight chains.
- *Our algorithms are particular to the motion constraints of planar, hexagonal modules.* Our algorithms are distinguished from many other planners such as [3,33,39,42] by the shape of the modules and the constraints on module movement we consider.

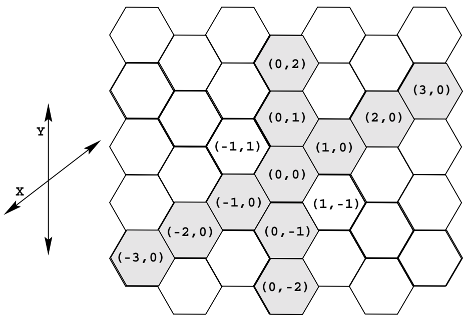


Fig. 4. Coordinates in a system of hexagonal cells

Outline: In Sect. 2 we describe the system assumptions.

Section 3 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 (i.e., are collinear) and intersect in one cell.

The result for collinear intersecting I and G is not applicable to all non-collinear intersecting straight chain configurations of I and G. Since one of our goals is to identify surfaces that can be traversed without collision or deadlock in a system of hexagonal robots, we also provide algorithms for all possible orientations of intersecting straight chains. Extensions to generalize the collinear algorithm when I and G are straight chains that occur in any initial relative intersecting orientations are presented in Sect. 4.

In Sect. 5 we present lower bounds on the number of moves and time required for the reconfiguration algorithms.

Section 6 provides a discussion of our results and future work.

2 System model

2.1 Coordinate system

The plane is partitioned into equal-sized hexagonal cells, which are arranged in a lattice and labeled using the coordinate system presented in [7], as shown in Fig. 4.

Given the coordinates of two cells, $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, the *lattice distance* [7], LD, between them is defined 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 is the minimum number of moves a module would need to move from cell c_1 to cell c_2.

2.2 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:

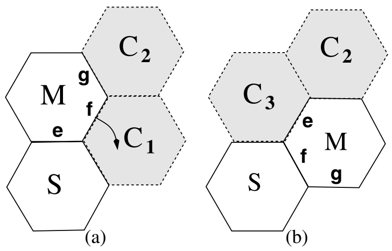


Fig. 5. Before (a) and after (b) module movement: M is moving, S is substrate, C_1, C_2, and C_3 are empty cells

  – its location (the coordinates of the cell that it currently occupies) [2],
  – its orientation (which edge is facing in which direction)[3], and
  – which of its neighboring cells is occupied by another module.

Modules move according to the following rules.

- Modules move in synchronous 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 round[4] (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 Fig. 5 for an example.
- Only one module tries to move into a particular cell in each round.[5]
- Modules do not carry other modules.

3 Collinear algorithm (Case 0)

In this section, we develop a distributed reconfiguration algorithm for a particular configuration of the system described in Sect. 2. 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 module. Our algorithm is always-connected and goal-stopping.

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 presented by Murata et al. in [17], which is based on vertex contacts. In Fig. 6 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*

[2] Initial coordinates could be determined using current GPS technology, broadcast to modules via wireless transmission, and recalculated individually as modules move.

[3] Initial orientation information could be broadcast to modules via wireless transmission and recalculated individually as modules move.

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

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

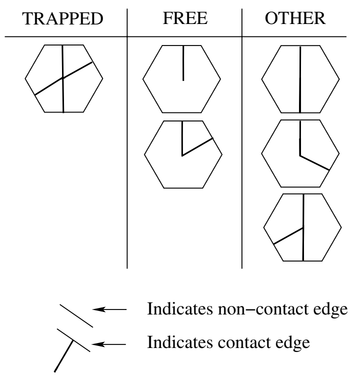


Fig. 6. Configuration types possible in collinear algorithm

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 Sect. 2.2). Modules classified as *free* are required to move in our algorithm. The *other* category includes modules whose movements are not allowed by our algorithm because it enforces the always-connected property. Even though the movement of modules in the *other* category would not violate hardware constraints, such movement might cause the configuration to become disconnected.

3.1 Collinear reconfiguration algorithm

3.1.1 Algorithm assumptions

- Each module knows the total number of modules in the system, n, and the coordinates of all goal cells in G.
- Initially, one module is in each cell of I.
- I and G are collinear and overlap in exactly one cell.
- I is composed of a single connected component.

3.1.2 Overview of algorithm

The algorithm works in synchronous rounds. Initially, each module calculates whether it will be moving clockwise (CW) or counter clockwise (CCW). A module calculates its position in I and uses the parity of its distance (i.e., even or odd lattice distance) from the cell in which I and G overlap to decide direction of movement.

Modules can determine locally from their initial contact pattern if they are part of a straight chain of modules and any module initially detecting a contact pattern that is inconsistent with a straight chain configuration can abort the reconfiguration in a pre-processing phase. Modules can calculate the position of G from the coordinates of the cells in G, information they receive during a pre-processing phase.

The algorithm instructs modules initially in positions with even distance from the overlap to rotate CCW and those with odd distance to rotate CW. Once a module calculates its rotation direction, it continues to rotate in that direction throughout the execution. In each round, each module calculates whether it is free (cf. Fig. 6) and moves if it is free in the direction calculated initially. Modules in G do not move.

3.1.3 Data structures at each module

- *contacts*: Boolean array indicating on which edges a module has 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.

The collinear reconfiguration algorithm is shown in Fig. 7.

```
Code for each module $\notin G$:
Initially:  // find movement direction
   1.   if (($n$ − $LD$($myCoord$, $goalCell$[1])) is even)
   2.        $d = CCW$
   3.   else  $d = CW$
In round $r = 1, 2, \ldots$ :  // move
   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)
```

Fig. 7. Pseudocode for collinear reconfiguration algorithm. $LD(c_1, c_2)$ is the lattice distance between cells c_1 and c_2

In Fig. 8 we depict an execution of the collinear reconfiguration algorithm when $n = 4$. For purposes of analysis, modules are labeled 1 through 4. Nine rounds are required for this reconfiguration.

3.2 Analysis of reconfiguration algorithm

All the results in this section, Theorems 1–3, are based on the assumptions of Sect. 3.1.1. In particular, the cost analysis in Theorem 3 relies on I and G overlapping in only one cell. When the overlap is greater than one cell, the cost decreases as shown in Corollary 1.

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, $1 \leq i \leq n$. Figure 9 shows a snapshot during an execution of the reconfiguration algorithm, where modules retain the labels of their initial cell designation. We will refer to a cell's neighboring cells as north (N), northeast (NE), southeast (SE), south (S), southwest (SW), and northwest (NW). We refer to the column of cells containing I and G as the *central* column and to the columns of cells to the east and west of the central column as the *outer* columns.

Theorem 1. *The collinear reconfiguration algorithm is correct.*

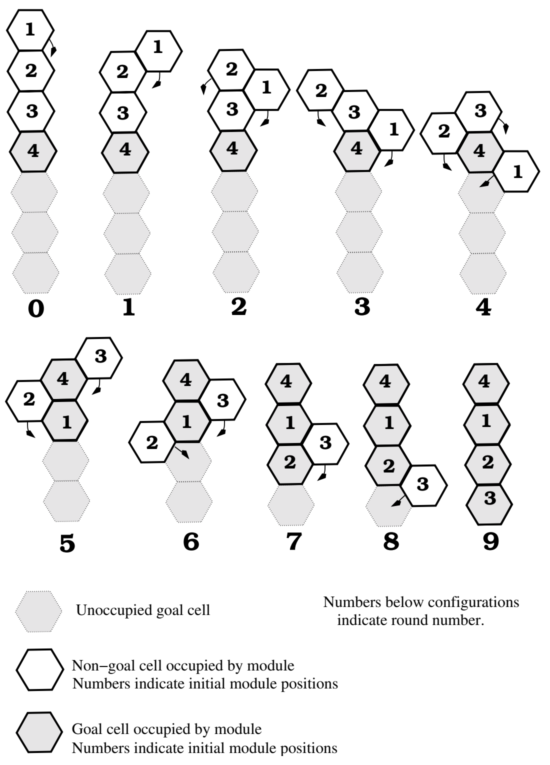

0
1
2
3
4
5
6
7
8
9
Unoccupied goal cell
Numbers below configurations indicate round number.
Non−goal cell occupied by module
Numbers indicate initial module positions
Goal cell occupied by module
Numbers indicate initial module positions

Fig. 8. Example collinear reconfiguration

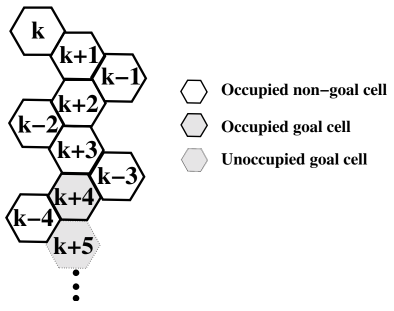


Fig. 9. Configuration at end of round $2k-1$, k is even

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 9 illustrates the configuration at the end of round $2k-1$.

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 cells in the outer columns. These contacts will be non-contiguous, so module i will not be free in round S. Referring to Fig. 9, 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 Fig. 9, 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 $I2$ 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 $i+j$. Referring to Fig. 9, module $k-2$ is in a position like that described for module i at the end of round $S-1$.

Case 3: $S \geq 2i-1+n$.
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 nature 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 $i-2, i-3, \ldots, 1$ 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 Fig. 9, 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. □

Theorem 2. *The collinear algorithm is goal-stopping and always-connected.*

Proof. The algorithm is goal-stopping because modules in G do not run the pseudocode in Fig. 7, and therefore do not move in any round.

The proof of Theorem 1 shows that, during execution of the collinear reconfiguration algorithm, modules in the outer columns move over a substrate of connected modules in the center column. It follows from the arguments about the positions of modules during execution of this algorithm that no intermediate configuration is partitioned. Therefore, the algorithm maintains the always-connected property. □

Theorem 3. *The collinear reconfiguration algorithm takes* $3(n-1)$ *rounds and makes* n^2-1 *module movements.*

Proof. Invariants $I1$ through $I3$, from the proof of Theorem 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. □

The proof of Theorem 3 can be used to prove the following corollary for variable-sized overlap between I and G.

Corollary 1. *When there is an overlap of size h in the modules of I and the positions of G, the algorithm takes* $(n-h)(n+1)$ *moves and* $2(n-h)+n-1$ *rounds.*

4 Non-collinear chain algorithms

We have developed extensions to the collinear reconfiguration algorithm to allow the reconfiguration of an initial chain to an overlapping final chain that may intersect the initial chain in any orientation. These extensions also do not require any message-passing communication between modules, but use counting techniques combined with knowledge of goal positions to determine the final position of each module.

4.1 Non-collinear chain reconfiguration algorithms

4.1.1 Algorithm assumptions

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

- I is oriented in a north-south fashion,
- G is oriented in a "southwest-northeast" fashion,
- the northernmost module in I is not in G,
- if there are cells in I on both the north and south sides of the intersection of I and G, then there are goal cells on the east side of I,
- if there are cells in I on both the north and south sides of the intersection of I and G and cells in G on both the east and west sides of I, then the north segment of I is at least as long as the west segment of G, and
- as in Sect. 3.1.1, 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.

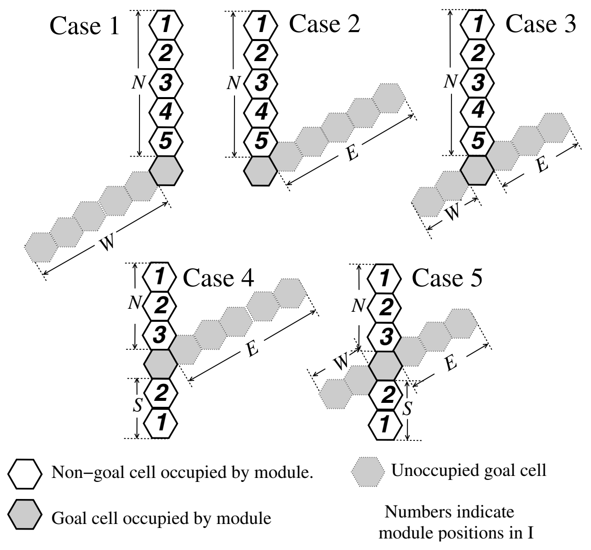

Case 1
Case 2
Case 3
Case 4
Case 5
N
W
E
S
1
2
3
4
5
Non−goal cell occupied by module.
Unoccupied goal cell
Goal cell occupied by module
Numbers indicate module positions in I

Fig. 10. Cases for calculating rotation direction and delay

4.1.2 General chain reconfiguration cases

The cases for non-collinear chain-to-chain reconfiguration are shown in Fig. 10. Note that if a particular orientation of intersecting chains I and G does not satisfy assumptions 1–5, the coordinates can be flipped horizontally and/or vertically so that the assumptions are satisfied. For example, in Fig. 10, the variant of case 4 with G slanted to the SW instead of NE of I can be obtained from the case 4 shown by a horizontal and a vertical inversion.

Throughout the remainder of this paper, we will refer to the numbers of the modules in I shown in Fig. 10 as module *positions* in I. As depicted in Fig. 10, the variable $\mathcal{N}$ (resp., $\mathcal{S}$) refers to the set of positions in I that are north (resp., south) of the goal cells and the variable $\mathcal{E}$ (resp., $\mathcal{W}$) refers to the set of positions in G that are east (resp., west) of the cells in I.

4.1.3 Overview of algorithms

The number of possible configuration types is larger in the general case than it was in the collinear case, as shown in Fig. 11. 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.

As in the collinear case, modules can determine locally from their initial contact pattern if they are part of a straight chain of modules during a pre-processing phase. Modules calculate the position of G using the coordinates of cells in G, information they receive during a pre-processing phase. Note that the initial coordinates of I and its intersection with G could be determined by message passing, but in this approach, modules would still need to discover the exact coordinates of G through additional inter-module communication. We avoid the necessity of inter-module message passing by assuming modules are given the coordinates of G during pre-processing (e.g., through a broadcast wireless transmission from a base station.)

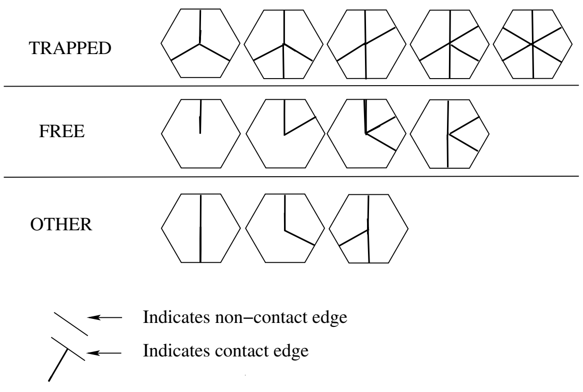


Fig. 11. Configuration types possible in general chain cases

```
Code for module $\notin G$:
Initially  $numW = numE = 0$.

• In round $r = 1, 2, \ldots$ prior to module's first move:
  if in $\mathcal{N}$:
   • if N cell is occupied or newly vacated, then
     – if NW cell is newly occupied, then $numW$++
     – if NE cell is newly occupied, then $numE$++
   • if N cell is newly vacated, then
     – calculate $direction$ and $delay$ using $numW$ and $numE$

  if in $\mathcal{S}$:
   • if S cell is occupied or newly vacated, then
     – if SW cell is newly occupied, then $numW$++
     – if SE cell is newly occupied, then $numE$++
   • if S cell is newly vacated, then
     – calculate $direction$ and $delay$ using $numW$ and $numE$

• In round $r = 1, 2, \ldots$:
  if $isFree$():
       – if $delay = 0$ then move $direction$
       – else $delay$−−

Procedure $isFree$():
   1.   $flips$ = 0
   2.   for ($i = 0$ to 5) do
   3.       if ($contacts$[$i$] $\neq$ $contacts$[($i$ + 1) % 6])
   4.           $flips$++
   5.   return (($flips$ == 2) and ($contacts$[$i$] < 5))
```

Fig. 12. Non-collinear chain reconfiguration schema

The algorithms for cases 1 through 5 are similar to that for the collinear case. The differences are the way modules

- determine themselves to be free,
- calculate the *direction* in which to move, and
- calculate the *delay*, i.e., how long to wait after becoming free until beginning to move.

The *isFree*() procedure presented in Sect. 3 must be modified to take into account the increased number of configuration types available (cf. Fig. 11). The modified version is included along with the algorithm schema in Fig. 12. Modules calculate direction and delay by counting modules with lower initial positions as they pass. The progress made on filling in goal cells in $\mathcal{E}$ and $\mathcal{W}$ is determined locally by maintaining separate tallies for modules passing on the east and west side. In this pseudocode, a cell c is *newly vacated* in round i ($i \geq 1$) if it is vacant in round i and it was occupied in round $i-1$. A cell c is *newly occupied* in round i ($i \geq 1$) if it is occupied in round i and it was vacant in round $i-1$.

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

1. avoid collisions and
2. avoid deadlock (a situation where the module is not free but not in a goal position).

We define the following patterns of rotation for a module in position i, $i > 1$. These patterns will be used in the presentation of our reconfiguration algorithms for cases 1–5:

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

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.

The rotation and delay patterns for cases 0 through 2 of chain-to-chain reconfiguration are given in Fig. 13. For cases 1 and 2, we present two algorithms, one that conserves rounds (*round-conserving*) and another that conserves moves (*move-conserving*). For completeness, we include the collinear reconfiguration algorithm, case 0.

The rotation and delay patterns for cases 3 through 5 chain-to-chain reconfiguration are given in Fig. 14. It should be noted that the two entries for case 5 in Fig. 14 are due to the fact that in the first case listed, all modules can rotate toward an obtuse angle and, in the second case, some of the modules must rotate toward an acute angle.

4.2 Analysis of non-collinear chain reconfiguration algorithms

The correctness of the non-collinear chain reconfiguration algorithms can be shown with an inductive argument like that used in the proof of Theorem 1. We omit these proofs in this paper because they are straightforward.

We analyze the running time and number of total moves in Theorems 4 through 8. These results are summarized in Tables 2 and 3 in Sect. 5.

Theorem 4. *The round-conserving case 1 reconfiguration algorithm takes* $3(n-1)$ *rounds and makes* $n^2 - 2$ *module movements if* n *is even and takes* $3(n-1)$ *rounds and makes* $n^2 - 1$ *moves if* n *is odd.*

Proof. In a case 1 reconfiguration, the last module to reach its designated goal cell is in position $n-2$ (see Fig. 15, parts (a) and (b)). The module in position 1 starts moving in round 1 and modules 2 through $n-1$ start moving 2 rounds after their north neighbor begins moving. The last module to stop moving in the execution is the module initially in position $n-2$. This module begins moving in round $1 + 2(n-3)$ and moves in $n+2$ additional rounds after its first move, resulting in a total of $2n - 5 + n + 2 = 3n - 3 = 3(n-1)$ rounds.

For even n (see Fig. 15(a)), module 1 moves n spaces, modules in even positions move $n+3$ spaces each, modules

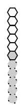
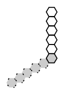
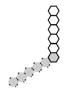
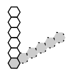
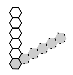
Case	n	Position	Pattern of rotation & delay
0 	any	1	*CW, delay = 0*
		2 to n–1	*bidirectional, delay = 0*
1  round–conserving	odd	1	*CW, delay = 0*
		2 to n–1	*bidirectional, delay = 0*
	even	1	*CCW, delay = 0*
		2 to n–1	*bidirectional, delay = 0*
1  move–conserving	any	1	*CCW, delay = 0*
		2 to n–1	*unidirectional, delay = 1*
2  round–conserving	any	1	*CW, delay = 0*
		2 to n–1	*bidirectional, delay = 1*
2  move–conserving	any	1	*CW, delay = 0*
		2 to n–1	*unidirectional, delay = 2*

Fig. 13. Rotation and delay by module position in I for cases 0 through 2. In configurations shown in column 1, goal cells are shaded and occupied cells have dark borders. Module positions are as shown in Fig. 10

in odd positions > 1 move $n-1$ spaces each, and the module in position n moves 0 spaces. So the total number of moves is

$$\begin{aligned} & n + \left(\frac{(n-2)}{2} \cdot (n-1)\right) + \left(\frac{(n-2)}{2} \cdot (n+3)\right) \\ &= n + \frac{(n^2-3n+2)}{2} + \frac{(n^2+n-6)}{2} \\ &= n + (n^2-n-2) \\ &= n^2-2. \end{aligned}$$

For odd n (see Fig. 15(b)), modules in even positions move $n+3$ spaces each, modules in odd positions move $n-1$ spaces each, and the module in position n moves 0 spaces. So the total number of moves is

$$\begin{aligned} & \left(\frac{(n-1)}{2} \cdot (n-1)\right) + \left(\frac{(n-1)}{2} \cdot (n+3)\right) \\ &= \frac{(n^2-2n+1)}{2} + \frac{(n^2+2n-3)}{2} \\ &= n^2-1. \end{aligned}$$

□

Theorem 5. *The move-conserving case 1 reconfiguration algorithm takes* $4(n-1)-2$ *rounds and makes* n^2-n *module movements.*

Proof. The module rotation directions for the case 1 move-conserving algorithm are shown in Fig. 15(c). Because of the constraints on module movement and the way the modules determine themselves to be free, when modules in case 1 all rotate CCW toward the obtuse angle bend, each module in I (in position > 1) must wait 3 rounds after its north neighbor begins moving prior to its first move. The module in position $n-1$ (the last module to stop moving) begins moving in round $1+3(n-2)$ and moves in $n-1$ additional rounds. When all modules rotate CCW, this is also the last module to stop moving, resulting in a total of $(3n-5)+(n-1) = 4n-6 = 4(n-1)-2$ rounds for the execution.

Each of $n-1$ modules move n spaces each, resulting in n^2-n module movements. □

Theorem 6. *The round-conserving algorithm for case 2 uses* $4(n-1)+1$ *rounds and* $n^2-\frac{n}{2}-\frac{1}{2}$ *moves for odd* n, *and* $4(n-1)-1$ *rounds and* $n^2-\frac{n}{2}-1$ *moves for even* n.

Proof. Refer to Fig. 16(a) and (b). In case 2 reconfigurations, each module in I in a position > 1 waits 3 rounds to move after its north neighbor begins moving. For odd n, (Fig. 16(b)), the module in position $n-1$ is the last module to reach its final goal position. Module $n-1$ starts moving in round $1+3(n-2) = 3n-5$ and takes an additional $n+2$ rounds. Therefore, the number of rounds used in such an execution is $3n-5+n+2 = 4n-3 = 4(n-1)+1$. For even n (Fig. 16(a)), the module in position $n-2$ is the last module to reach its final goal position. Module $n-2$ starts moving in round $1+3(n-3) = 3n-8$ and takes an additional $n+3$ rounds. Therefore, the number of rounds in such an execution is $3n-8+n+3 = 4n-5 = 4(n-1)-1$.

For the number of moves, we need to consider how many moves are used by each module in I. For odd n, the module in position 1 makes $n-1$ moves, the module in position $n-1$ makes $n+3$ moves, the $\frac{n-3}{2}$ modules in even positions 2 through $n-2$ make $n+4$ moves each, and the $\frac{n-3}{2}$ modules in odd positions 3 through $n-2$ make $n-3$ moves each. This

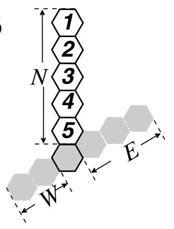
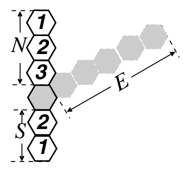
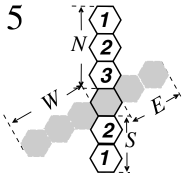
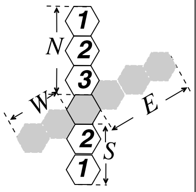
Case	Set	Position	Pattern of rotation & delay
3 	*N*	1	*Toward acute angle, delay = 0*
		2 to |E|	*unidirectional, delay = 2*
		|E|+1	*Toward obtuse angle, delay = 0*
		|E|+2 to n−1	*unidirectional, delay = 1*
4 	*S*	1	*Toward obtuse angle, delay = 0*
		2 to |S|	*unidirectional, delay = 1*
	N	1	*Toward acute angle, delay = (3*|S|) – |N| + 1*
		2 to |N|	*bidirectional, delay = 1*
5  *|N| = |W|*	*S*	1	*Toward obtuse angle, delay = 0*
		2 to |S|	*unidirectional, delay = 1*
	N	1	*Toward obtuse angle, delay = 0*
		2 to |N|	*unidirectional, delay = 1*
5  *|N| > |W|*	*S*	1	*Toward obtuse angle, delay = 0*
		2 to |S|	*unidirectional, delay = 1*
	N	1	*Toward acute angle, delay = (3*|S|) – |N| + 1*
		if (|E|–|S|) > 1 2 to (|E|–|S|)	*unidirectional, delay = 2*
		(|E|–|S|) + 1	*Toward obtuse angle, delay = 0*
		(|E|–|S|)+2 to n−1	*unidirectional, delay = 1*

Fig. 14. Rotation and delay by module position in I for cases 3 through 5. In configurations shown in column 1, goal cells are shaded gray, occupied cells have dark borders, and sets $\mathcal{N}$, $\mathcal{S}$, $\mathcal{E}$, and $\mathcal{W}$ are labeled

results in

$$\begin{aligned}
&(n-1)+(n+3)+\left(\frac{(n-3)}{2}\cdot(n+4)\right)+\left(\frac{(n-3)}{2}\cdot(n-3)\right)\\
&=2n+2+\left(\frac{(n^2+n-12)}{2}\right)+\left(\frac{(n^2-6n+9)}{2}\right)\\
&=\frac{2n^2-n+1}{2}\\
&=n^2-\frac{n}{2}+\frac{1}{2}
\end{aligned}$$

module movements. For even n, the module in position 1 makes $n-1$ moves, the module in position $n-1$ makes $n-2$ moves, the $\frac{n-2}{2}$ modules in even positions 2 through $n-2$ make $n+4$ moves each, and the $\frac{n-4}{2}$ modules in odd positions 3 through $n-3$ make $n-3$ moves each. This results in

$$\begin{aligned}
&(n-1)+(n-2)+\left(\frac{(n-2)}{2}\cdot(n+4)\right)+\left(\frac{(n-4)}{2}\cdot(n-3)\right)\\
&=2n-3+\left(\frac{(n^2+2n-8)}{2}\right)+\left(\frac{(n^2-7n+12)}{2}\right)
\end{aligned}$$

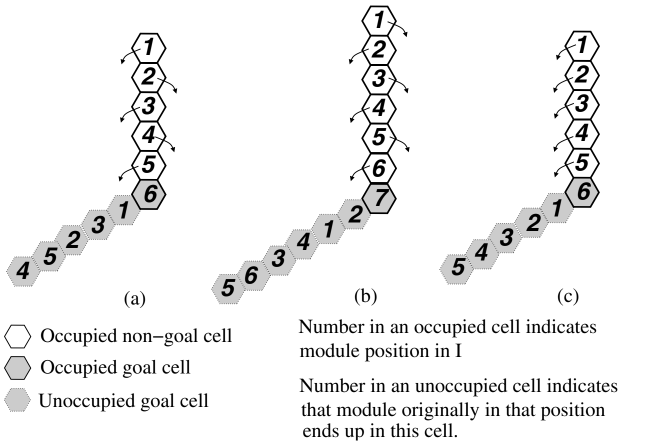

(a)
(b)
(c)
Occupied non−goal cell
Occupied goal cell
Unoccupied goal cell
Number in an occupied cell indicates module position in I
Number in an unoccupied cell indicates that module originally in that position ends up in this cell.

Fig. 15. Round-Conserving case 1 rotations for even (a) and odd (b) n. Move-conserving case 1 rotations (c)

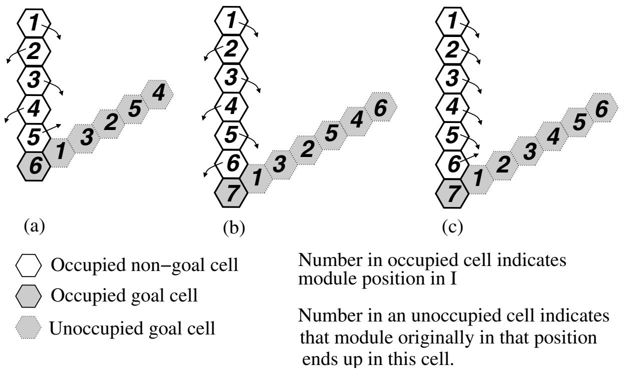

(a)
(b)
(c)
Occupied non–goal cell
Occupied goal cell
Unoccupied goal cell
Number in occupied cell indicates module position in I
Number in an unoccupied cell indicates that module originally in that position ends up in this cell.

Fig. 16. Round-Conserving case 2 rotations for even (a) and odd (b) n. Move-conserving case 2 rotations (**c**)

$$= \frac{2n^2 - n - 2}{2}$$
$$= n^2 - \frac{n}{2} - 1$$

module movements. □

Theorem 7. *The move-conserving algorithm for case 2 uses* $n^2 - 3n + 4$ *moves and* $5(n-2)+1$ *rounds.*

Proof. Consider part (c) of Fig. 16. To get to its final position, module 1 makes $n-1$ moves. Module $n-1$ also makes $n-1$ moves. Module n makes 0 moves. The remainder of the modules make $n-2$ moves each. This gives us

$$2(n-1) + (n-3)(n-2)$$
$$= 2n - 2 + n^2 - 5n + 6$$
$$= n^2 - 3n + 4$$

module moves.

Since all modules are moving CW toward the acute angle bend and because moving modules must be separated by 2 empty cells to avoid deadlock in the corner, each module in I in position > 1 must wait 4 rounds after its north neighbor begins moving prior to its first move. The module in position $n-1$ is the last module to start and the last to stop moving, as shown in Fig. 16(c). This module begins moving in round $1+4(n-2)$ and moves in $n-2$ additional rounds. This results in a total of $4n-7+n-2 = 5n-9 = 5(n-2)+1$ rounds. □

Theorem 8. *Cases 3 through 5 use* $\Theta(n^2)$ *module movements and take* $\Theta(n)$ *rounds.*

Proof. Theorem 8 follows from the fact that the reconfiguration algorithms for cases 3 through 5 are combinations of case 1 and 2 reconfiguration algorithms.

Case 3 uses the move-conserving case 2 algorithm with modules rotating toward the acute angle to fill the goal cells in $\mathcal{E}$ and uses the move-conserving case 1 algorithm with modules rotating toward the obtuse angle to fill the goal cells in $\mathcal{W}$. Since one of $\mathcal{W}$ and $\mathcal{E}$ is at least $\frac{n}{2}$ while the other is at most $\frac{n}{2}$, this result follows from Theorems 5 and 7.

Case 4 begins with the move-conserving case 1 algorithm for the modules in $\mathcal{S}$ and ends with the round-conserving case 2 algorithm for modules in $\mathcal{N}$. In this case, one of $\mathcal{N}$ and $\mathcal{S}$ is at least $\frac{n}{2}$ while the other is at most $\frac{n}{2}$, so this result follows from Theorems 5 and 6.

In case 5, if $|\mathcal{N}| = |\mathcal{W}|$, the reconfiguration uses the move-conserving case 1 algorithm for modules in both $\mathcal{N}$ and $\mathcal{S}$. If $|\mathcal{N}| > |\mathcal{W}|$ in case 5, the modules in $\mathcal{S}$ use the move-conserving case 1 algorithm and the modules in $\mathcal{N}$ start with the move-conserving case 2 algorithm and end with the move-conserving case 1 algorithm. One of $\mathcal{N}$ and $\mathcal{S}$ is at least $\frac{n}{2}$ while the other is at most $\frac{n}{2}$ and one of $\mathcal{W}$ and $\mathcal{E}$ is at least $\frac{n}{2}$ while the other is at most $\frac{n}{2}$. Therefore, the number of moves and rounds used follows from Theorems 5 and 7. □

Theorem 9. *The algorithms presented for cases 1 through 5 are goal-stopping and always-connected.*

Proof. The algorithms are goal-stopping because modules in G do not run the pseudocode in Fig. 12, and therefore do not move in any round. A proof similar to the ones for Theorems 1 and 2 can be given to show that these algorithms maintain the always-connected property. □

Simple extensions to the algorithms allow the reconfiguration of a chain in any initial location to a chain in any final location by calculating intermediate configurations that lead to eventual intersection and then running one of the algorithms described in this section. To calculate intermediate configurations for non-intersecting I and G, project a line through the center of some module in I along the NW-SE, SW-NE, or N-S axis so that the line intersects the center of some cell in G. If necessary, use one of the non-collinear chain reconfiguration algorithms to reorient I so that it is collinear to this projected line and then use the collinear chain reconfiguration algorithm to move I so that it intersects G. After I and G intersect, choose one of the chain reconfiguration algorithms to complete the reconfiguration process.

4.2.1 Non-goal-stopping, round-conserving algorithms for reconfiguration in cases 3 and 5

For cases 3 and 5, we can conserve rounds if the algorithm does not require all modules to stop when they reach a goal position. Figure 17 shows the positions of I and G for case 3 (part (a)) and for case 5 when $|\mathcal{N}| > |\mathcal{W}|$ (part (b)). In these cases, modules in $\mathcal{N}$ rotating CCW cut through the darker shaded goal cell to fill in goal cells in $\mathcal{E}$, allowing modules to move bidirectionally and therefore increasing parallelism. This slight modification to the algorithms allows modules in $\mathcal{N}$ to start out using the round-conserving case 2 algorithm. These algorithms use $\Theta(n)$ rounds and $\Theta(n^2)$ moves as did the other algorithms for these cases. Even though this strategy does not change the asymptotic bounds, it improves the constant hidden in the asymptotic notation, especially if $\mathcal{E} > \mathcal{W}$ and $\mathcal{N} > \mathcal{S}$.

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 by Chirikjian and Pamecha in [8]. We show a larger lower bound on the number of moves required for a collinear (case 0) reconfiguration when the algorithm is goal-stopping and we show this bound is tight. For

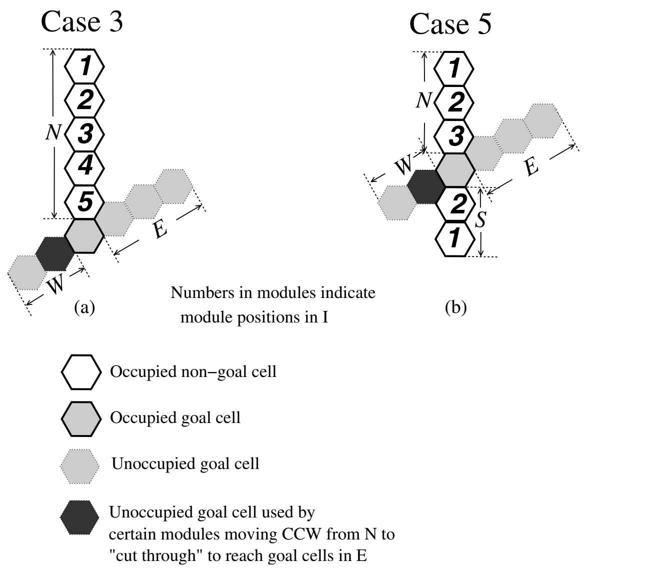

Case 3
Case 5
N
E
W
S
1
2
3
4
5
(a)
(b)
Numbers in modules indicate module positions in I
Occupied non–goal cell
Occupied goal cell
Unoccupied goal cell
Unoccupied goal cell used by certain modules moving CCW from N to "cut through" to reach goal cells in E

Fig. 17. Configurations for non-goal-stopping algorithms for cases 3 (a) and 5 (b)

case 1 goal-stopping algorithms, we show that the lower bound given in [8] is tight and we show the bounds given in [8] are asymptotically tight for goal-stopping algorithms in cases 2 through 5.

In the second part of this section, we consider the lower bound for the total elapsed time of reconfiguration, a measure that was not addressed in [8]. Our bounds are asymptotically tight in this measure.

5.1 Lattice distance sum

In [23], Pamecha and Chirikjian show that a key concept for the lower bounds on the number of moves used during reconfiguration is that of the *initial lattice distance sum*, which we denote as D_0. For given initial and goal configurations, let D_0 be the minimum, over all choices for the locations of the modules in the goal configuration, of the sum, over all modules i, of the lattice distance between i's initial position and i's final position. In Theorems 10, 11, and 12, we derive specific D_0 values for each of the chain reconfiguration cases.

As shown below, the lower bounds for both the number of moves and the number of rounds used during reconfiguration are functions of D_0, the initial lattice distance sum.

Theorem 10. *For cases 0 and 1,* $D_0 = n^2 - n$.

Proof. Number the cells in the initial configuration I from 1 to n and the cells in the goal configuration G from n to $2n-1$. Let v_i be the lattice distance between module i's initial position, cell i, and its final position, cell g_i, for $1 \le i \le n$. For both case 0 and case 1, this lattice distance is $g_i - i$. Note that g_1 through g_n form a permutation of n through $2n-1$. Thus

$$D_0 = \sum_{i=1}^{n} v_i = \sum_{i=1}^{n} (g_i - i) = n^2 - n.$$

□

Theorem 11. *For case 2,* $D_0 = (n^2 - n)/2$.

Proof. Number the cells in the initial configuration I from 1 to n. The shortest distance between module $n-i$ and any goal cell is i, $1 \le i \le n-1$. Thus

$$D_0 = \sum_{i=1}^{n-1} (n-i) = (n^2 - n)/2.$$

□

Theorem 12. *For cases 3 through 5,* $D_0 = \Omega(n^2)$.

Proof. Referring to Fig. 10, for case 3, Theorems 10 and 11 imply that D_0 is at least

$$(|\mathcal{W}|^2 - |\mathcal{W}|) + \frac{(|\mathcal{E}|^2 - |\mathcal{E}|)}{2}.$$

For case 4, the implication is that D_0 is at least

$$(|\mathcal{S}|^2 - |\mathcal{S}|) + \frac{(|\mathcal{N}|^2 - |\mathcal{N}|)}{2}.$$

For case 5, D_0 is at least

$$\frac{(|\mathcal{N}|^2 - |\mathcal{N}|)}{2} + \frac{(|\mathcal{S}|^2 - |\mathcal{S}|)}{2}.$$

Since one of $\mathcal{W}$ and $\mathcal{E}$ is at least $\frac{n}{2}$ while the other is at most $\frac{n}{2}$, and similarly for $\mathcal{N}$ and $\mathcal{S}$, the theorem follows. □

5.2 Number of module movements

The following theorem is a result of Chirikjian and Pamecha [8]:

Theorem 13. *Any algorithm to reconfigure a metamorphic system under the system assumptions given must cause at least* D_0 *module movements.*

For a goal-stopping algorithm to reconfigure a collinear chain (case 0), we show that the lower bound is larger than D_0 from Theorem 10.

Theorem 14. *Any goal-stopping algorithm to reconfigure a metamorphic system under the system assumptions given must cause at least* $n^2 - 1$ *module movements in case 0.*

Proof. Note that the module initially in position n never moves, since it starts in a goal cell and the algorithm is goal-stopping. Thus, each of the remaining $n-1$ modules must make an additional move out of the main column to go around that module. Adding $n-1$ to the $n^2 - n$ bound from Theorem 10 produces a lower bound of $n^2 - 1$. □

The lower bound from Theorem 14 matches the number of moves taken by our case 0 algorithm (cf. Theorem 2), and therefore this bound is tight for goal-stopping algorithms.

For case 1 reconfiguration, the number of moves used by the move-conserving algorithm is $n^2 - n$ (cf. Theorem 5). This matches the lower bound in Theorem 10, and therefore the lower bound for case 1 is tight.

Table 2. Upper and lower bounds on number of module moves for chain-to-chain reconfiguration algorithms. All algorithms are goal-stopping

Reconfiguration Algorithm	Lower Bound	Upper Bound
Case 0	if goal-stopping $n^2 - 1$	$n^2 - 1$
	if not goal-stopping $n^2 - n$	
Case 1	$n^2 - n$	$n^2 - n$
Case 2	$\frac{(n^2-n)}{2}$	$n^2 - 3n + 4$
Case 3	$\Theta(n^2)$	$\Theta(n^2)$
Case 4	$\Theta(n^2)$	$\Theta(n^2)$
Case 5	$\Theta(n^2)$	$\Theta(n^2)$

Recall the number of moves for the move-conserving algorithm for case 2 is $n^2 - 3n + 4$ (cf. Theorem 7). Comparing this to the $\frac{n^2-n}{2}$ bound from Theorem 11, we see that this bound is asymptotically tight.

For cases 3 through 5, the number of moves taken by modules executing each of our algorithms is $\Theta(n^2)$ (cf. Theorem 8). From Theorem 12, we can see that this bound is asymptotically tight.

5.2.1 Summary of bounds on number of module movements

Table 2 shows the upper and lower bounds on the number of module moves using D_0 for each case from Sect. 5.1.

5.3 Number of rounds

We now show lower bounds on the number of rounds required for reconfiguration.

Theorem 15. *Any algorithm to reconfigure a metamorphic system in case 0 under the system assumptions given must take at least* $\frac{D_0}{\lfloor \frac{1}{2}n \rfloor}$ *rounds.*

Proof. Clearly, at the end of the reconfiguration, the lattice distance between each module's current and final position is zero. Note that, in each round, each module can decrease its lattice distance to its final position by at most one.

Claim. The maximum number of processors that can decrease their lattice distance to their final position in one round in case 0 under the system assumptions given is $\lfloor \frac{n}{2} \rfloor$.

Proof of claim: The maximum number of modules that can move in one round is $\lfloor \frac{3}{4}n \rfloor$. To see why, consider that a single substrate module S has 6 sides and at most 6 neighbors. The constraints on module movement require each moving module to move into an unoccupied cell. Therefore, 3 out of every 6 sides surrounding S must be unoccupied in order for 3 modules to move over S. We argue below that in case 0, at most two out of every four modules can decrease their lattice distance to their final position during a single round.

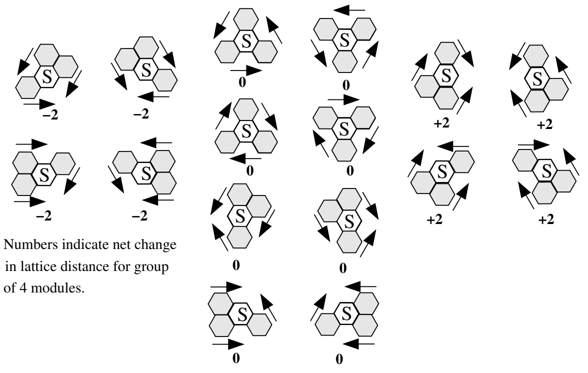

S
−2
−2
−2
−2
0
0
0
0
0
0
0
0
+2
+2
+2
+2
Numbers indicate net change in lattice distance for group of 4 modules.

Fig. 18. Possible moves of three modules over one substrate (S)

Figure 18 shows an exhaustive listing of all possible moves of three modules over one substrate, along with the net change in lattice distance (assuming I and G are aligned as in case 0) below each figure. In this figure, moving modules are shaded and the unshaded cell marked S is occupied by a non-moving substrate module. To see that this listing is exhaustive, consider the 6 sides of a module (where the sides are numbered consistently) as a binary string, where if a side is connected to a neighbor, it has a 1 in that binary digit and a 0 otherwise. There are 20 possible combinations of three 0's and three 1's in this list of binary strings. However, all three 1's (or 0's) cannot be contiguous, since this would correspond to the case where a module moves into a cell that was occupied during the previous round. This restriction excludes 6 of these 20 combinations. Of the 14 possible combinations remaining, 2 can be represented twice (with modules rotating in opposite directions), resulting in the 16 configurations shown. Since the largest net decrease for any group of modules moving over one stationary module is 2, the claim follows. □

By Theorem 13, at least D_0 moves are needed to reconfigure. Thus the number of rounds needed in case 0 is at least $\frac{D_0}{\lfloor \frac{1}{2}n \rfloor}$. □

Theorem 16. *Any algorithm to reconfigure a metamorphic system in cases 1 through 5 under the system assumptions given must take at least* $\frac{D_0}{\lfloor \frac{3}{4}n \rfloor}$ *rounds.*

Proof. Because at most $\lfloor \frac{3}{4}n \rfloor$ modules can move in one round due to assumptions on module movement, at most $\lfloor \frac{3}{4}n \rfloor$ modules can decrease their lattice distance to their final position during that round in cases 1 through 5. □

5.3.1 Algorithms that do not maintain the always-connected property

For the collinear (case 0) reconfiguration, we can obtain an upper bound that is closer to the lower bound on the number

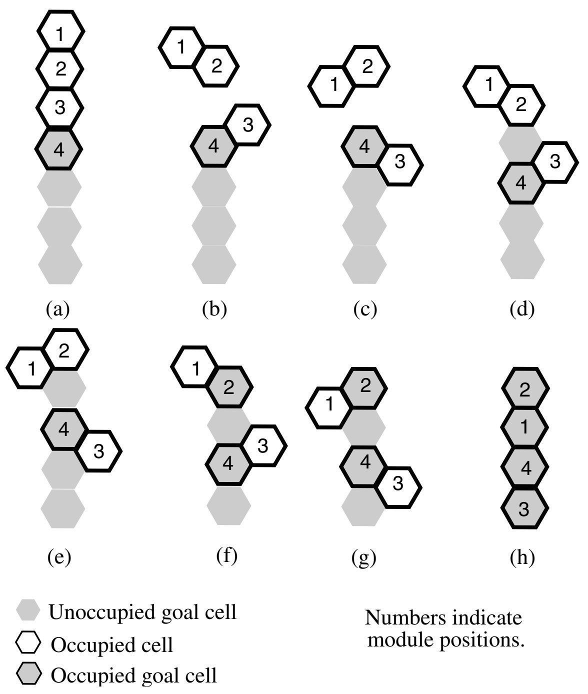


Fig. 19a–h. Snapshots during execution of collinear reconfiguration algorithm when n is even and algorithm does not obey goal-stopping and always-connected properties. Time is progressing from part (a) to (h)

of rounds if we do not require that the algorithm maintains the always-connected property. Here is an algorithm that does not maintain the always-connected property when n is even:

1. In rounds 1 and 2, modules 1, 5, 9, ... rotate CCW and modules 3, 7, 11, ... rotate CW, with module 1 using module 2 for a substrate, module 3 using module 4 for a substrate, and so on.
2. In round 3, modules 2, 6, 10, ... rotate CW using modules 1, 5, 9, ... as substrates, and modules 4, 8, 12, ... rotate CCW using modules 3, 7, 11, ... as substrates.
3. In round 4, modules 1, 5, 9, ... rotate CCW using modules 2, 6, 10, ... as substrates, and modules 3, 7, 11, ... rotate CW using modules 4, 8, 12, ... as substrates.
4. Repeat step 2 in odd-numbered rounds and step 3 in even-numbered rounds until modules with even position numbers are in goal positions $n-2$ spaces south of their original positions.
5. In the last two rounds, modules 1, 5, 9, ... rotate CCW and modules 3, 7, 11, ... rotate CW into goal positions.

Figure 19 depicts an execution of the above reconfiguration algorithm for case 0 when the algorithm is not required to be goal-stopping or always-connected and when n is even. Part (a) is a snapshot of the initial configuration and parts (b)–(h) are snapshots of the first 7 rounds of the execution.

When n is odd, we cannot break the modules evenly into moving pairs, as we did for the case when n is even. So modules 1 to 3 move to their goal positions in a connected group of three modules.

1. Modules 1 through 3 run the case 0 goal-stopping algorithm, beginning with module 1 rotating in the CW direction.

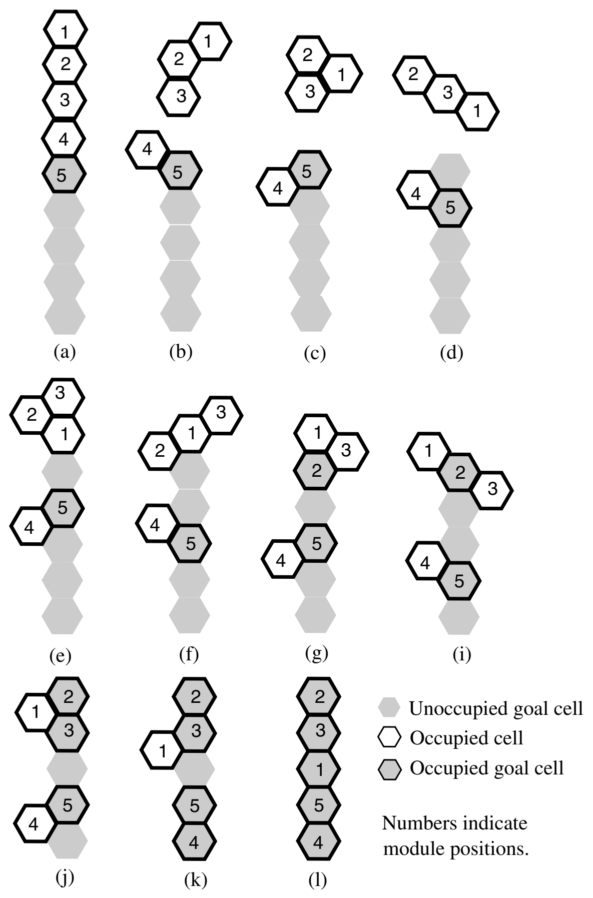


Fig. 20a–k. Snapshots during execution of collinear reconfiguration algorithm when n is odd and algorithm does not obey goal-stopping or always-connected properties. Time is progressing from part (a) to (k)

2. In rounds 1 and 2, modules 4, 8, 12, ... rotate CCW using modules 5, 9, 13, ... as substrates and modules 6, 10, 14, ... rotate CW using modules 7, 11, 15, ... as substrates.
3. In round 3, modules 5, 9, 13, ... rotate CW using modules 4, 8, 11, ... as substrates, and modules 7, 9, 13, ... rotate CCW using modules 6, 8, 12, ... as substrates.
4. In round 4, modules 4, 8, 12, ... rotate CW using modules 5, 9, 13, ... as substrates and modules 6, 10, 14, ... rotate CCW using modules 7, 11, 15, ... as substrates.
5. Repeat step 3 in odd-numbered rounds and step 4 in even-numbered rounds until modules 5, 7, 9, ... are in the goal cell $n-2$ spaces south of their initial positions, modules 4, 8, 12, ... are in the cell NW of the goal cell n spaces south of their original position, and modules 6, 10, 14, ... are in the cell NE of the goal cell n spaces south of their original position.
6. In the last round, modules 4, 8, 12, ... rotate CCW and modules 6, 10, 14, ... rotate CW into goal positions.

Figure 20 depicts an execution of the above reconfiguration algorithm for case 0 when the algorithm is not required to be always-connected and when n is odd. Part (a) is a snapshot of the initial configuration and parts (b)–(k) are snapshots of the first 10 rounds of the execution.

When n is even, two rounds are required for modules in odd positions to move into place as substrates for even modules

Table 3. Upper and lower bounds on number of rounds for chain-to-chain reconfigurations

Reconfiguration Algorithm	Lower Bound	Upper Bound
Case 0	$2(n-1)$	if always-connected $3(n-1)$
		if not always-connected $2n-1$ n even; $2n$ n odd
Case 1	$\frac{4}{3}(n-1)$	$3(n-1)$
Case 2	$\frac{2}{3}(n-1)$	$4n-5$ n even; $4n-3$ n odd
Case 3	$\Omega(n)$	$\Theta(n)$
Case 4	$\Omega(n)$	$\Theta(n)$
Case 5	$\Omega(n)$	$\Theta(n)$

in the outer columns on the east and west sides of the goal column. Modules in even positions move $n-2$ spaces at a rate of 1 space every 2 rounds after the first move, beginning in round 3 and arriving in their goal positions in round $2n-3$. Two additional rounds are required for modules in odd positions to arrive in their goal positions, so the running time for the algorithm is $2n-1$. This comes within one round of matching the lower bound given in Theorem 15.

When n is odd, two rounds are required to get modules in even positions ≥ 4 into position as substrates for modules in odd positions ≥ 4 in the outer columns. After this, even modules ≥ 4 move $n-2$ spaces at a rate of 1 space every 2 rounds. One final move puts even modules in even positions ≥ 4 into their goal positions. Since modules in odd positions > 4 take less time to arrive in their goal positions, this gives us $2+2(n-2)+1 = 2n-1$ rounds for all modules in positions ≥ 4 to reach goal positions. Module 1 takes 4 rounds to move to position 4. After this, modules 1 through 3 advance by one module in the central column every 2 rounds for $n-2$ more rounds. This gives us $4+2(n-2) = 2n$ rounds for the running time in this case.

5.3.2 Summary of bounds on number of rounds

Table 3 contains the upper and lower bounds on the number of rounds using D_0 for each case of the chain-to-chain reconfiguration.

While the number of rounds used by our algorithms do not match exactly the lower bounds presented in Table 3, these bounds are asymptotically tight in all cases. We conjecture that the upper bounds for algorithms that do not maintain the always-connected property may more closely match the lower bounds in cases 1 through 5, just as they did in case 0.

6 Conclusions and future work

The algorithms presented in this paper rely on total knowledge of the goal configuration. Additionally, 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, 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 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 showed that the collinear algorithm that does not maintain the always-connected property is optimal in the number of rounds used for even n and nearly optimal for odd n. We conjecture that algorithms better matching the lower bounds can be found for cases 1 through 5 if the always-connected property is not maintained.

The chain-to-chain reconfiguration algorithms presented in this paper have served as building blocks for the development of reconfiguration algorithms for more arbitrary goal configurations. In our follow-up work to the chain-to-chain reconfigurations, we have developed algorithms to accomplish the distributed reconfiguration of robot chains into simple "admissible" goal configurations [35,36]. Our latest work focuses on developing algorithms to perform the reconfiguration from chains to "admissible" configurations efficiently [34].

The goal of our future work is to develop more complex distributed reconfiguration strategies from building blocks like the ones 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.

Acknowledgements. We thank the anonymous reviewers of an earlier version of this paper for helpful comments.

References

1. Bojinov H, Casal A, Hoag T: Emergent structures in modular self-reconfigurable robots. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, vol. 2, pp 1734–1741 (2000)
2. Bojinov H, Casal A, Hoag T: Multiagent control of self-reconfigurable robots. In: Proc. of Fourth Intl. Conf. on Multi-agent Systems, pp 143–150 (2000)
3. Butler Z, Byrnes S, Rus D: Distributed motion planning for modular robots with unit-compressible modules. In: Proc. of IEEE Intl. Conf. on Intelligent Robots and Systems, pp 790–796 (2001)
4. Casal A, Yim M: Self-reconfiguration planning for a class of modular robots. In: Proc. of SPIE Symposium on Intelligent

Systems and Advanced Manufacturing, vol. 3839, pp 246–256 (1999)

5. Castano A, Shen W-M, Will P: CONRO: Towards miniature self-sufficient metamorphic robots. Autonomous Robots Journal 8: 309–324 (2000)
6. Chiang C-J, Chirikjian G: Similarity metrics with applications to modular robot motion planning. Autonomous Robots Journal, special issue on self-reconfiguring robots, 10(1): 91–106 (2001)
7. Chirikjian G: Kinematics of a metamorphic robotic system. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 449–455 (1994)
8. Chirikjian G, Pamecha, A: Bounds for self-reconfiguration of metamorphic robots. Proc. IEEE International Conference on Robotics and Automation, pp 1452–1457 (1996)
9. Chirikjian G: Metamorphic hyper-redundant manipulators. In: Proc. of Intl. Conf. on Advanced Mechatronics, pp 467–472 (1993)
10. Chirikjian G, Pamecha A, Ebert-Uphoff I: Evaluating efficiency of self-reconfiguration in a class of modular robots. Journal of Robotic Systems 13(5): 317–338 (1996)
11. Dumitrescu A, Suzuki I, Yamashita M: High speed formations of reconfigurable modular robotic systems. Proc. IEEE International Conference on Robotics and Automation, pp 123–128 (2002)
12. Fukuda T, Buss M, Hosokai H, Kawauchi Y: Cell structured robotic system CEBOT (control, planning and communication methods). Intelligent Autonomous Systems 2: 661–671 (1989)
13. Hamlin GJ, Sanderson AC: Tetrobot: A modular approach to reconfigurable parallel robotics. Kluwer Academic Publishers, Newton, MA (1997)
14. Hosokawa K, Tsujimori T, Fujii T, Kaetsu H, Asama H, Kuroda Y, Endo I: Self-organizing collective robots with morphogenesis in a vertical plane. In: IEEE Intl. Conf. on Robotics and Automation, pp 2858–2863 (1998)
15. Kotay K, Rus D: Scalable parallel algorithm for configuration planning for self-reconfigurable robots. In: Proc. of the Conf. on Sensor Fusion and Decentralized Control in Robotic Systems III, (SPIE RB06) (2000)
16. Lee WH, Sanderson AC: Dynamic analysis and distributed control of the tetrobot modular reconfigurable robot system. Autonomous Robots Journal, special issue on self-reconfiguring robots 10(1): 67–82 (2001)
17. Murata S, Kurokawa H, Kokaji S: Self-assembling machine. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 441–448 (1994)
18. Murata S, Kurokawa H, Tomita K, Kokaji S: Self-assembling method for mechanical structure. In: Artif. Life Robotics 1: 111–115 (1997)
19. Murata S, Kurokawa H, Yoshida E, Tomita K, Kokaji S: A 3-D self-reconfigurable structure. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 432–439 (1998)
20. Murata S, Yoshida E, Kamimura A, Kurokawa H, Tomita K, Kokaji S: M-TRAN: Self-reconfigurable modular robotic system. IEEE/ASME Trans. on Mechatronics 7(4): 431–441 (2002)
21. Nguyen A, Guibas LJ, Yim M: Controlled module density helps reconfiguration planning. New Directions in Algorithmic and Computational Robotics. A.K. Peters, pp 23–36 (2001)
22. Nilsson M: Free climbing snake robot. IEEE Control Systems, pp 21–26 (1998)
23. Pamecha A, Chirikjian G: A useful metric for modular robot motion planning. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 442–447 (1996)
24. Pamecha A, Ebert-Uphoff I, Chirikjian G: Useful metrics for modular robot motion planning. IEEE Trans. on Robotics and Automation 13(4): 531–545 (1997)
25. Pamecha A, Chiang C-J, Stein D, Chirikjian G: Design and implementation of metamorphic robots. In: Proc. of ASME Design Engineering Technical Conf. and Computers in Engineering Conf. (1996)
26. Prevas K, Unsal C, Efe M, Khosla P: A hierarchical motion planning strategy for a uniform self-reconfigurable module robotic system. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 787–792 (2002)
27. Rus D, Vona M: Physical implementation of the crystalline robot. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 1726–1733 (2000)
28. Rus D, Vona M: Crystalline robots: Self-reconfiguration with compressible unit modules. Autonomous Robots Journal, special issue on self-reconfigurable robots 10(1): 107–124 (2001)
29. Salemi B, Shen W-M, Will P: Hormone-controlled metamorphic robots. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, pp 4194–4199 (2001)
30. Suh J, Homans S, Yim M: Telecubes: Mechanical design of a module for self-reconfigurable robotics. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp 4095–4101 (2002)
31. Tomita K, Murata S, Kurokawa H, Toshida E, Kokaji S: Self-assembly and self-repair method for a distributed mechanical system. IEEE Trans. on Robotics and Automation 15(6): 1035–1045 (1999)
32. Unsal C, Kiliccote H, Khosla P: A modular self-reconfigurable bipartite robotic system: implementation and motion planning. Auton. Robot. 10: 23–40 (2001)
33. Vassilvitskii S, Yim M, Suh J: A complete, local and parallel reconfiguration algorithm for cube style modular robots. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp 117-122 (2002)
34. Walter J, Tsai B, Amato N: Choosing good paths for fast distributed reconfiguration of hexagonal metamorphic robots. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp 102–109 (2002)
35. Walter J, Welch J, Amato N: Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. IEEE Trans. on Robotics and Automation (2002,) to appear
36. Walter J, Welch J, Amato N: 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, 4196: 441–453 (2000)
37. Yim M: A reconfigurable modular robot with many modes of locomotion. In: Proc. of Intl. Conf. on Advanced Mechatronics, pp 283–288 (1993)
38. Yim M, Duff D, Roufas K: Polybot: a modular reconfigurable robot. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp 514–520 (2000)
39. Yim M, Lamping J, Mao E, Chase JG: Rhombic dodecahedron shape for self-assembling robots. SPL TechReport P9710777, Xerox PARC (1997)
40. Yoshida E, Murata S, Tomita K, Kurokawa H, Kokaji S: Distributed formation control of a modular mechanical system. In: Proc. of the Intl. Conf. on Intelligent Robots and Systems, pp 1090–1097 (1997)
41. Yoshida E, Murata S, Kurokawa H, Tomita K, Kokaji S: A distributed reconfiguration method for 3-D homogeneous structure. In: Proc. of the Intl. Conf. on Intelligent Robots and Systems, pp 852–859 (1998)
42. Zhang Y, Yim M, Lamping J, Mao E: Distributed control for 3D shape metamorphosis. Autonomous Robots Journal, Special Issue on Self-Reconfigurable Robots (2000)