Optimally Efficient Goal-Filling Algorithms for Self-Reconfigurable Hexagonal Metamorphic Robots

Jamee Bateau Elianne Schutze Jennifer E. Walter
Department of Computer Science
Vassar College
{jabateau,elschutze,jewalter}@vassar.edu

Abstract

The problem addressed is the distributed reconfiguration of a system of two-dimensional mobile robots (modules) from an initial straight chain into an arbitrary goal configuration that satisfies simple admissibility conditions. Our reconfiguration strategy depends on finding a contiguous path of cells, called a substrate path, that spans the goal configuration. Modules fill in this substrate path and then move along the path to fill in the remainder of the goal without collision or deadlock.

We present algorithms that improve the efficiency and enlarge the set of goal configurations that can be filled by the algorithms presented in [14] and [15], our previous work. Our algorithms combine techniques used in the goal-filling algorithms with bridging algorithms presented in [6]. We developed two strategies for efficient goal filling and compared their performances to existing goal-filling algorithms as well as to each other via simulation using a discrete event simulator. We implemented an algorithm for finding a substrate path that could more evenly bisect the goal configuration, and an algorithm for optimally filling a substrate path, developed in previous unpublished work. The results of our simulation are presented and discussed.

1 Introduction

A *self-reconfigurable* robotic system is a collection of independently controlled, mobile robots, each of which has the ability to connect, disconnect, and move around adjacent robots. *Metamorphic* robotic systems [3], a subset of self-reconfigurable systems, are further limited by requiring each module to be identical in structure, motion constraints, and computing capabilities. Typically, the modules have a regular symmetry so that they can be packed densely, i.e., packed so that gaps between adjacent modules are as small as possible. In these systems, robots achieve locomotion by moving over a substrate composed of one or more other robots. The mechanics of locomotion depend on the hardware and can include module deformation to crawl over neighboring modules [4, 10] or to expand and contract to slide over neighbors [11]. Alternatively, moving robots may be constrained to rigidly maintain their original shape, requiring them to roll over neighboring robots [7, 17, 18].

Shape changing in these composite systems is envisioned as a means to accomplish various tasks, such as bridge building, structural support, satellite recovery, or tumor excision [10]. The complete interchangeability of the robots provides a high degree of system fault tolerance. Also, self-reconfiguring robotic systems are potentially useful in environments that are not amenable to direct human observation and control (e.g., interplanetary space, undersea depths).

The motion planning problem for a metamorphic robotic system is to determine a sequence of robot motions required to go from a given initial configuration (I) to a desired goal configuration (G).

Most of the existing motion planning strategies rely on centralized algorithms to plan and supervise the motion of the system components [4, 5, 10, 11, 16]. Others use distributed approaches which rely on heuristic approximations or require communication between robots in each step of the reconfiguration process [1, 7, 8, 17, 18].

We focus on a system composed of planar, hexagonal robotic modules as described by Chirikjian [4]. We consider a distributed motion planning strategy, given the assumption of initial global knowledge of G. Our distributed approach offers the benefits of localized decision making and the potential for greater system fault tolerance. Additionally, our strategy requires less communication between modules than other approaches. We have previously applied this approach to the problem of reconfiguring a straight chain to an intersecting straight chain [13] and a straight chain to a goal configuration that satifies a general "admissibility" condition [12]. In these papers, a centralized algorithm was described for determining whether an arbitrary goal configuration is admissible.

2 Related work

Chirikjian [4] and Pamecha [10] discuss centralized algorithms for planar hexagonal modules that use the distance between all modules in I and the coordinates of each goal position to accomplish the reconfiguration of the system. Pamecha et al. [10] define the distance between configurations as a metric and apply this metric to system self-reconfiguration using a simulated annealing technique to drive the process towards completion.

Centralized motion planning strategies for systems of two dimensional robotic modules are also examined by Nguyen

et al. [9] and analysis is presented for the number of moves necessary for specific reconfigurations.

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

Distributed approaches are taken by Murata, et al. to reconfigure a system of two dimensional hexagonal modules [7], and a system of three dimensional cubic modules [8]. Yim et al. [17] and Zhang et al. [18] present distributed algorithms to reconfigure three dimensional rhombic dodecahedral modules. Each of these algorithms are probabilistic and require substantial message passing between neighboring modules.

Our approach

This paper examines distributed motion planning strategies for a planar metamorphic robotic system undergoing a reconfiguration from a straight chain to a goal configuration satisfying certain properties. In our algorithms, robots are identical, but act as independent agents, making decisions based on their current position and the sensory data obtained from physical contacts with adjacent robots. Our purpose is to seek an understanding of the necessary building blocks for reconfiguration, starting with algorithms in which no messages need to be passed between participating robots during reconfiguration. Reconfiguration in certain scenarios, like the ones presented in this and our earlier papers [12, 13], can be accomplished using algorithms that do not require any message passing. Therefore, our algorithms are more communication efficient than the distributed approaches of [1, 7, 17] and [18].

In this paper, we consider two dimensional, hexagonal robots like those described by Chirikjian [3]. Our proposed scheme uses a classification of robot types based on connected edges similar to the classification used by Murata et al. [7] for connected vertices. In the algorithms presented in this paper, each robot independently determines whether it is in a movable state based on the cell it occupies in the plane, the locations of cells in the goal configuration, and on which sides it contacts neighbors. Robots move from cell to cell and modify their states as they change position. Since the robots know the coordinates of the goal cells, we show that each of them can independently choose a motion plan that avoids module collision.

3 System model

Assumptions about modules

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

Our model provides an abstraction of the hardware features and the interface between the hardware and the application layer.

- Each module is identical in computing capability and runs the same program.
- Each module is a hexagon of the same size as the cells of the plane and always occupies exactly one of the cells.
- Each module knows at all times:
  - its location (the coordinates of the cell that it currently occupies),
  - its orientation (which edge is facing in which direction), and
  - which of its neighboring cells is occupied by another module.

Modules move according to the following rules.

1. Modules move in lockstep rounds.
2. In a round, a module M is capable of moving to an adjacent cell, C_1, iff (see Fig. 1 for an example)
   (a) cell C_1 is currently empty,
   (b) module M has a neighbor S that does not move in the round (called the *substrate*) and S is also adjacent to cell C_1, and
   (c) the neighboring cell to M on the other side of C_1 from S, C_2, is empty.
3. Only one module tries to move into a particular cell in each round.

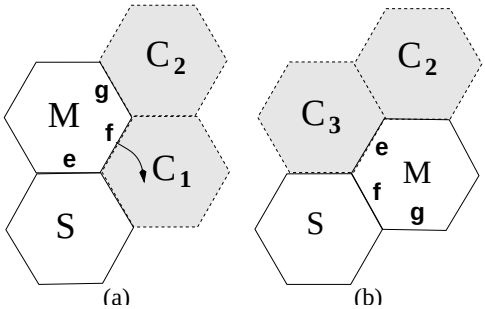

Figure 1: Before (a) and after (b) module movement: M is moving, S is substrate, and C_1, C_2, and C_3 are empty cells.

If the algorithm does not ensure that each moving module has an immobile substrate, as specified in rule 2(b), then the results of the round are unpredictable. Likewise, the results of the round are unpredictable if the algorithm does not ensure rule 3.

4 Problem definition

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

Definition 1 *An admissible goal configuration has no holes and no vertical gaps.*

5 Finding substrate paths

Our goal was to find a substrate path that most evenly bisected the goal configuration so as to maximize the number of goal-cells being filled at a time. An evenly bisected goal allowed modules to bidirectionally break off the initial chain and simultaneously filled both north and south of the substrate path. If the substrate path found was not continuous, we used the path algorithm presented in [14] instead. The procedure for finding an admissible substrate path in G proceeds in two steps:

1. Determine the midpoint cell in each column of the goal configuration and assign it to be a path cell. If a column length is even, the chosen path cell would be the cell north of where the midpoint would be if the column length was odd.
2. Check if path is continuous. If path is continuous, use for substrate path. Otherwise, use path algorithm presented in [14] to find substrate path.

Once the substrate path is found, the moving modules fill the path first, and the remaining modules travel north and south of the substrate path to fill the rest of the goal configuration.

6 Bridges

Since we wanted single-cell spacing between moving modules, we encountered deadlock problems when the substrate path met a column in the goal at an acute angle. We solved this by placing a *bridge* in cells where deadlock would occur. A bridge is a moving module that is stopped or temporarily delayed a number of time steps to help the other moving modules reach their goal cells.

There are two different bridge types, permanent and temporary. Say Column A is being filled, and the column to the west of A is Column B.

- When there are goal cells in Column B, a permanent bridge is placed in the first goal cell of Column B and is stopped because it has reached its corresponding goal cell. The bridge is then used as a stepping stool for the remaining modules to fill the rest of Column A.
- When there are no goal cells in Column B, a temporary bridge is placed in the first cell (which is a non-goal cell) of Column B and is delayed for however many time steps are needed for all but the last cell of Column A to be filled. That temporary bridge cell then moves again and fills the last cell in Column A.

These bridging methods are used for every column in the goal configuration until the whole goal is filled.

7 Distributed reconfiguration

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

7.1 Algorithm assumptions

1. Each module knows the total number of modules in the system, n, and the goal configuration, G.
2. Initially, one module is in each cell of I.
3. G is an admissible configuration.
4. I and G overlap in one goal cell in column G_1.

7.2 Overview of algorithm

The algorithm works in synchronous rounds. In each round, each module determines whether it is free (cf. Fig. 2). In this figure, the modules labeled *trapped* are unable to move due to hardware constraints and those labeled *free* represent modules that are allowed to move in our algorithm, possibly after some initial delay. The modules in the *other* category are restricted from moving by our algorithm, not by hardware constraints.

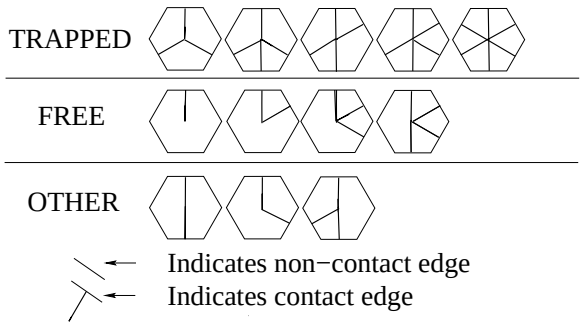

Figure 2: Contact patterns possible in algorithm.

Only module 0 (the module at the free end of I) can initially determine the exact time when it will begin moving. Other modules in I rely on local contact information to calculate their position in I and any possible delay after they

become *free* to avoid collision and deadlock. Once a module begins moving, it has only the local information about contacts with adjacent modules and its current coordinates to guide its part of the entire system reconfiguration.

All modules except module 0 dynamically calculate their position in I, direction of rotation, possible delay and final coordinates in G by counting the modules in initial positions further from the intersection of I and G as they pass, noting the direction (CW or CCW) in which the passing modules rotate. The module intersecting G does not move.

Let p be the array of coordinates of goal cells on the substrate path (stored locally at each module), starting with the cell that has an edge incoming from the cell in which I and G intersect in column G_1. Coordinates of goal cells to the north and south of the substrate path are also stored in arrays at each module. A module calculates the goal cell it will occupy using its position in I, the length of the arrays of coordinates on, north, and south of the substrate path, and the current count of modules that have passed on both sides.

Modules fill in the substrate path first. After every goal cell in p is filled, modules alternate rotation directions, filling the columns projecting north and south of p from east, G_m, to west, G_1.

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

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

The reconfiguration schema uses the shape of the substrate path to determine how to start and proceeds as follows:

- For those modules filling in the substrate path:
  - If the substrate path has no vertical segments, either modules 0 through $|p|-1$ or modules 0 through $|p|-2$ and $|p|+2$ use *(0,0)-bidirectional* pattern.
  - If p has vertical segments, modules $0 \ldots |p|-1$ use the *2-unidirectional* pattern in CW direction. Module $|p|$ begins *(0,0)-bidirectional* pattern, moving CCW (unless there are no cells to be filled in the CCW direction, in which case it continues the *1-unidirectional* pattern).
- For modules in positions $> |p|$ (i.e., these modules climb over the substrate path to fill the rest of G):
  - Modules use *(0,0)-bidirectional* pattern until all cells either north or south of p are filled. After this, modules use *1-unidirectional* pattern, with either CW or CCW direction.
  - Each module stops in the goal cell to the north or south of the substrate path that it has calculated it should occupy.
- Once a module stops for a round in a goal cell, it never moves out of that goal cell.

Local variables at each module include:

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

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

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

Figure 3: Pseudocode for all modules from straight chain to admissible G.

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

8 Conclusions and future work

We have presented a bridging algorithm for optimally filling an arbitrary goal configuration column by column that resolves collision and deadlock situations when they occur. We also considered an algorithm that allowed modules to fill a goal configuration layer by layer (i.e., row by row), but that has not been developed fully and will be left for future work. Once the layering algorithm is robust, we will be able

to compare the number of time steps taken for each algorithm to fill arbitrary goal configurations and determine which algorithm is more efficient.

References

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

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *Proc. of the ICRA 2001 Workshop on Modular Robots*, 2001.

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

[4] G. Chirikjian and A. Pamecha. Bounds for self-reconfiguration of metamorphic robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1452–1457, 1996.

[5] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule: design and control algorithms. In *Workshop on Algorithmic Foundations of Robotics*, pages 376–386, 1998.

[6] D. Little and J. Walter. Using Hexagonal Metamorphic Robots to Form Temporary Bridges. In Proc. of the IEEE International Conference on Intelligent Robotic Systems, Aug. 2005, Edmonton, Alberta, Canada, pages 2652-2657

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

[8] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 432–439, 1998.

[9] A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. To appear in *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.

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

[11] D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 2513–2520, 1999.

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

[13] J. Walter, J. Welch, and N. Amato. Distributed reconfiguration of metamorphic robot chains. In *Proc. of ACM Symp. on Principles of Distributed Computing*, pages 171–180, 2000.

[14] J. Walter, J. Welch, and N. Amato. Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. IEEE Transactions on Robotics and Automation, Vol. 18, No. 6, pp. 945-956, 2002.

[15] J. Walter, E. Tsai, and N. Amato. Algorithms for Fast Concurrent Reconfiguration of Hexagonal Metamorphic Robots. IEEE Transactions on Robotics, Vol. 21, No. 4, pp. 621-631, 2005.

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

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

[18] Y. Zhang, M. Yim, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. To appear in *Autonomous Robots Journal, special issue on self-reconfigurable robots*, 2000.