Distributed universal reconfiguration of 2D lattice-based modular robots

Ferran Hurtado* Enrique Molina* Suneeta Ramaswami† Vera Sacristán*

Abstract

We prove universal reconfiguration (i.e., reconfiguration between any two robotic systems with the same number of modules) of 2-dimensional lattice-based modular robots by means of a distributed algorithm. To the best of our knowledge, this is the first known reconfiguration algorithm that applies in a general setting to a wide variety of particular modular robotic systems, and holds for both square and hexagonal lattice-based 2-dimensional systems. All modules apply the same set of local rules (in a manner similar to cellular automata), and move relative to each other. Reconfiguration is carried out while keeping the robot connected at all times. The total number of time steps, moves and communication required for the reconfiguration is linear in the number of modules.

1 Introduction

1.1 Goal

We solve the following problem for 2-dimensional lattice-based modular robotic systems: Given two connected configurations with the same number of modules, reconfigure one into the other by means of a distributed algorithm. As far as we know, this is the first general reconfiguration algorithm encompassing both square and hexagonal regular lattices, and using a general framework that does not exploit specific characteristics of any particular robotic system. A large set of robotic prototypes fit within this framework.

In our framework, a robot is a connected configuration of homogeneous modules that are located in a 2-dimensional lattice. Each module can attach to and detach from a neighboring module, and can change its position to a neighboring empty grid position in the lattice by attaching to a neighboring module and moving with respect to it. Each module has constant size memory, can perform constant size computations, and can send or receive constant size messages to or from its neighboring modules. One designated module needs linear memory to store the information of the goal shape and to perform computations required for the reconfiguration algorithm.

Within this framework, our algorithm is completely distributed, local, and asynchronous. It consists of a set of rules, each one having a priority, a precondition, and an action or postcondition. Rules are identical for all modules, and are simultaneously executed by all of them in an asynchronous way. The term "local" here means that each module communicates with modules lying within a small neighborhood in order to execute the algorithm. In the procedure we propose, all modules know when they have reached their final destination.

1.2 Related work

Our approach builds on the seminal work of Beni [2], who proposed the conceptual model of cellular robotic systems, inspired by cellular automata. Since then, several authors have developed distributed algorithms for reconfiguring specific square lattice-based modular robot designs and shapes [8], as well as generic strategies for locomotion, reconfiguration and self-repairing for particular shapes [4, 10]. Simultaneously, locomotion and reconfiguration have been proved for some class of shapes within the hexagonal setting [6, 7, 15, 14, 9, 1], as well as for 3-dimensional lattices [3]. Local rules have also been used in the framework of a general metamodules' theory [5]. Recently, specific sets of rules have been proposed to produce reconfigurations between particular shapes of M-TRAN which are lattice-based [11]. To the best of our knowledge, this last work presents the first execution of a distributed local rules strategy on real robot units, hence proving its realizability beyond experimental simulation.

2 The model

In the square lattice setting, a module is any robotic unit located in a 2-dimensional square grid. We represent modules by squares occupying one grid cell, although their actual shape need not be a square. A module can independently attach to and detach from each of its 4 direct grid neighboring modules, if

*Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain. {ferran.hurtado,vera.sacristan}@upc.edu, Enrique-Molina@hotmail.com. F.H. and V.S. were partially supported by projects MTM2012-30951, MTM2009-07242, Gen. Cat. DGR 2009SGR1040, and ESF EUROCORES programme EuroGIGA, CRP ComPoSe: MICINN Project EUI-EURC-2011-4306, for Spain.

†Department of Computer Science, Rutgers University, Camden, U.S.A. rsuneeta@camden.rutgers.edu. Partially supported by NSF grant CCF-0830589.

present. A robot is a connected set of identical modules. By "connected" we mean that the adjacency graph of the robot configuration (a node in the center of each module and a straight line edge for each attachment among modules) is connected.

Modules cannot move on their own, but they can move relative to each other. To be more precise, a module may perform four relative motions, illustrated in Figure 1, where the dark colored module is performing the move. The first two moves are of the *change*

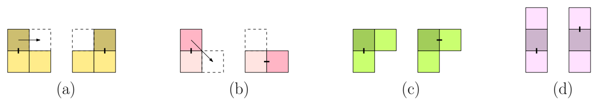

Figure 1: Change position moves: (a) slide (b) convex transition. Change attachments moves: (c) concave transition (d) opposite transition. All moves may apply in any of the 4 directions (N, S, E, W) relative to the moving module.

position type: a module performing *slide* or *convex transition* translates itself from its current lattice position to a neighboring one. The last two moves are of the *change attachments* type: a module performing *concave transition* or *opposite transition* changes its attachment from one neighbor to another without modifying its lattice position.

In our framework, the modules of the robot are indistinguishable, and each module is given and applies the same set of rules. In order to do so, we assume each module has a (simple) processor and some (small) memory, knows its own orientation (N, S, E, W) and state (active or passive), can detect whether it is attached to a neighbor, can send and receive (short) messages to and from neighbors, and is able to perform (elementary) operations with a few counters and text strings. For our reconfiguration algorithm, only one module needs to store the final configuration, which is a linear amount of information. This module, that we call the leader, can be either determined in advance or autonomously chosen by the set of modules [13].

As stated above, all modules run the same predefined set of rules. Each rule has the following structure: a priority, a precondition, and an action or postcondition. Priorities, represented as small integers, are used by the module to decide which of possibly several rules that apply to its situation is executed. A precondition is any constant size boolean combination of the following: compare priorities, check neighboring empty/filled positions, check own connections, match states/text or counters/integers, and compare calculation results with counters, messages and integers. A postcondition can be any *and* combination of the following: change position (slide, convex transition), change attachments (concave transition, opposite transition), modify state, compute and update counters, and send messages.

Notice that, as opposed to the square case, in a hexagonal lattice two modules cannot be vertex-adjacent: either they are edge-adjacent or they do not touch. This simplifies the moving rules, as there is no distinction between slide and convex transition. In the hexagonal case, the only move that modifies the position of a module consists of rotating around a static neighbor to move from one edge adjacency to a clockwise or counterclockwise neighboring position. For moves that change attachments, there is a wider range of choices than in the square lattice case.

In our model, changing position only requires the goal lattice position to be free. This assumption could be a potential limitation because the atomic robot units of several current prototypes need some extra empty space to produce slide and convex transition. Nevertheless, by appropriately grouping atomic robot units into meta-modules, we have been able to ensure that our moves can be safely made without extra free space requirements in three general models of reconfigurable robots: the expand/contract model, the sliding model, and the rotating model.

3 Overview of Reconfiguration Strategy

The solution we present is distributed because each module acts on its own without the need of a central controller, other than to get the reconfiguration process started. Our solution is parallel as all modules act in parallel. Our solution is local because each module only needs to communicate with modules within a small neighborhood when checking rule preconditions. In this context, the neighborhood of a module consists of all modules lying in grid positions within the second annulus around it. Finally, we should mention that our reconfiguration strategy (and our simulator) is intended to run in a synchronized framework. An asynchronous version can be obtained by means of a shaking hands strategy, at a cost of increased communication among the modules.

The overall strategy behind our algorithm is to move modules along the boundary of the robot to reconfigure in two stages. We first reconfigure the robot from its initial shape into a canonical shape (the strip configuration, in our case) and then reconfigure from the canonical to the final shape. The modules do not need to know the robot's complete initial shape. However, the goal shape needs to be known at least by the leader. In particular, our solution to reconfigure from the canonical to goal shape requires the leader to assign a final destination location for each module in the canonical configuration.

Our solution is based on the following general operating principles:

1. A particular spanning tree of the robot's adjacency graph, which we call the *scan tree*, is built so that all leaves of the tree lie on the boundary of the robot (see Figure 2). At the beginning, all modules are considered to be static. At any given instant, only leaf modules can start moving, i.e., go from static to active. Once a module is active, its node is considered to be cut from the spanning tree.

2. The movement of the active modules along the boundary of the static robot always follows the right hand rule (turn right along the robot boundary) when reconfiguring from the initial to canonical shape, and the left hand rule when reconfiguring from the canonical to goal shape.

3. Moving modules are not allowed to climb (move relative to) other moving modules. This is a reasonable assumption in order to avoid unbounded acceleration and unpredictable collisions.

4. Every module is assigned a number when constructing the above stated spanning tree. Generally speaking, this number corresponds to the DFS (depth first search) order numbering of the nodes of the scan tree of the initial shape for the initial to canonical reconfiguration, or the goal shape for the canonical to goal reconfiguration. This number is used to guide the moves of the modules and also to prove the correctness of our solution.

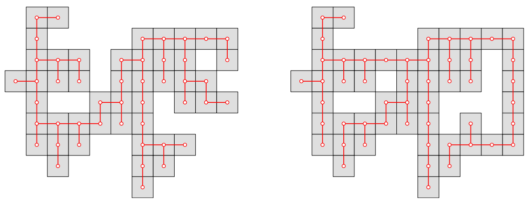
Figure 2: Left: the scan tree of a configuration without holes. Right: the scan tree of a configuration with holes.

Rules for advancing modules during reconfiguration must take care of various types of conflicts:

Activation conflicts occur when an active module tries to move to a position where it would attach to a static leaf which simultaneously becomes active. In this case, priority needs to be given either to the activation of the leaf or to the moving module. Any of the two choices is appropriate, as long as it stays consistent during the reconfiguration.

Collision conflicts occur when two active modules intend to move to the same lattice position. In this case, priority is given to the module with lower DFS order number.

Obstruction conflicts occur when an active module would like to move into a lattice position which is already occupied by an active module. In this case, deadlocked situations could be created if the boundary of the static shape forms bottlenecks. See Figure 3 for an illustration. We avoid deadlocks by means of specific "jumping rules" that have higher priority than the regular advancing ones. Jumping at a bottleneck allows a module to advance more than one position along the boundary of the static shape in one step to avoid entrance into the "cul-de-sac" region. Deciding which modules should jump and at which bottlenecks is crucial in our strategy. These decisions are made by comparing the DFS number of the static modules that form the bottleneck. The active module uses the DFS numbers to determine whether it is about to enter or exit the corresponding cul-de-sac. The reconfiguration from the strip to the goal shape essentially reverses the forward (initial shape to strip) reconfiguration procedure. Modules march from the rightmost end of the strip following the left hand rule and the goal shape is constructed in a clockwise depth-first manner. However, two additional issues need to be addressed in the reverse reconfiguration. The main difference between the forward and backward procedures is that modules in the strip must be sent to their final goal destination in the order given by the depth-first traversal of the final shape. Therefore, the jumping rules need to be modified to make sure that

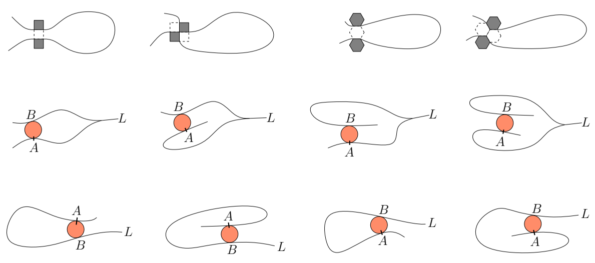

Figure 3: *First row:* The two possible bottleneck types in square lattices (left) and the hexagonal lattices (right). The continuous line schematizes the boundary of the robot shape. *Second and third rows:* The active module x (depicted as a red/dark disc) is attached to the static module A and decides whether to jump, i.e., attach to B and detach from A. The decision is based on whether B belongs to a branch in the tree previous, in DFS order, to that of A (second row) or the same branch as that of A (third row). This allows x to determine if it is about to enter or exit the cul-de-sac and consequently, whether or not to jump at the bottleneck. In both rows, L indicates the position of the root of the tree, namely the leader module. The continuous lines schematize the branches of the tree involved in the bottleneck A, B.

no jump changes the order of the active modules when they march along the boundaries of the static structure. In addition, it is also necessary to ensure that for robot shapes with holes, no active module gets trapped within the wrong hole or outside its destination hole because of premature closing of the hole during the reconfiguration procedure.

4 Main result

We have proved that, within a square or hexagonal lattice, it is possible to reconfigure any connected set of modules into any other connected set of the same number of modules by means of a distributed asynchronous algorithm based on local rules. If executed in a synchronous way, any reconfiguration of a robotic system of n modules is done in $O(n)$ time steps with $O(n)$ basic moves per module, using $O(1)$ force per module, $O(1)$ size memory and computation per module (except for one module, which needs $O(n)$ size memory to store the information of the goal shape), and $O(n)$ communication per module.

5 Simulations

We have implemented our rules [12] for square lattices in a synchronized simulator [13], and have applied them to a large set of reconfigurations (Figure 4 shows a screen shot). We are currently working on the hexagonal lattice simulations.

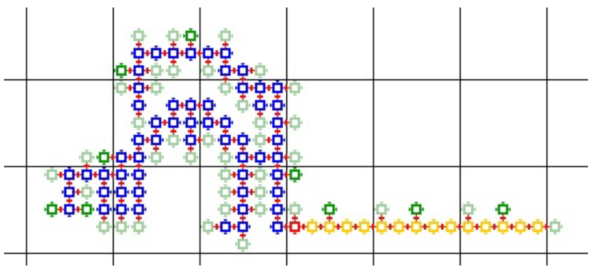

Figure 4: A screen shot of the simulation of a reconfiguration. Dark blue modules are static, green modules are active, and the yellow horizontal strip on the right is being formed.

References

[1] J. Bateau, A. Clark, K. McEachern, E. Schutze, and J. Walter. Increasing the efficiency of distributed goal-filling algorithms for self-reconfigurable hexagonal metamorphic robots. In *Proc. of the International Conference on Parallel and Distributed Techniques and Applications*, to appear, 2012.

[2] G. Beni. The concept of cellular robotic system. In *Proc. of the IEEE International Symposium on Intelligent Control*, pages 57–62, 1988.

[3] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1734–1741, 2000.

[4] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *Int. J. Robot. Res.*, 23:919–937, 2004.

[5] D. J. Dewey, M. P. Ashley-Rollman, M. De Rosa, S. C. Goldstein, T. C. Mowry, S. S. Srinivasa, P. Pillai, and J. Campbell. Generalizing metamodules to simplify planning in modular robotic systems. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1338–1345, 2008.

[6] A. Dumitrescu, I. Suzuki, and M. Yamashita. Formations for fast locomotion of metamorphic robotic systems. *Int. J. Robot. Res.*, 23(6):583–593, 2004.

[7] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Trans. Robot. Autom.*, 20(3), 2004.

[8] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2858–2863, 1998.

[9] P. Ivanov and J. Walter. Layering algorithm for collision-free traversal using hexagonal self-reconfigurable metamorphic robots. In *Proc. of IEEE/RSJ International Conference on Robots and Systems (IROS)*, pages 521–528, 2010.

[10] K. Kotay and D. Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2362–2369, 2004.

[11] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *Int. J. Robot. Res.*, 27:373–386, 2008.

[12] O. Rodríguez. Simulating distributed action of modular robots. Degree thesis, Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, 2013 (in Catalan).

[13] R. Wallner. A system of autonomously self-reconfigurable agents. Diploma thesis, Institute for Software Technology, Graz University of Technology, Graz, Austria, 2009.

[14] J. E. Walter, E. M. Tsai, and N. M. Amato. Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots. *IEEE Trans. Robot.*, 21(4):621–631, 2005.

[15] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17:171–189, 2004.