Reconfiguring sliding squares in-place by flooding*

Joel Moreno[1] **and Vera Sacristán**[1]

1 **Universitat Politècnica de Catalunya**
joel.moreno97@gmail.com, vera.sacristan@upc.edu

Abstract

We present a new algorithm that reconfigures between any two edge-connected configurations of n sliding squares within their bounding boxes. The algorithm achieves the reconfiguration by means of $\Theta(n^2)$ slide moves. A visual simulator and a set of experiments allows us to compare the performance over different shapes, showing that in many practical cases the number of slide moves grows significantly slower than in others as n increases.

1 Introduction

As defined in [13], "Modular self-reconfigurable (MSR) robots are robots composed of a large number of repeated modules that can rearrange their connectedness to form a large variety of structures. An MSR system can change its shape to suit the task, whether it is climbing through a hole, rolling like a hoop, or assembling a complex structure with many arms."

Their versatility, though, comes with a drawback: the complexity of the algorithms required to control MSR systems and make them walk, self-repair or, more generally, reconfigure. Reconfiguring modular robots without a thoughtful method can generate an excessive number of moves (and therefore too much time and power consumption), disconnections of the robot, collisions between its modules, dead-locks and other complex situations.

In this paper we propose a new and efficient in-place universal reconfiguration algorithm for a class of lattice-based modular robots.

1.1 Basic Definitions

A *connected robot configuration* (in short, a *robot* or a *configuration*) is any edge-connected set of squares (*modules* of the robot) located in a 2-dimensional square lattice. In other words, it is a polyomino. Within the *sliding-square* setting, robot modules (represented as squares) move relative to each other by sliding along their shared edges, as shown in Figure 1.

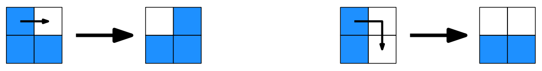


Figure 1 The sliding move allows straight transitions (left) as well as convex transitions (right).

A *hole* in a configuration is any vertex-connected component of empty lattice cells. A *pseudo-hole* is any edge-connected component of empty lattice cells. See Figure 2 for an illustration. Notice that, with this definitions, the unbounded component is considered to be a hole/pseudo-hole. Notice also that pseudo-holes are subsets of holes.

* V.S. was partially supported by MTM2015-63791-R (MINECO/FEDER) and Gen. Cat. DGR 2017SGR1640.

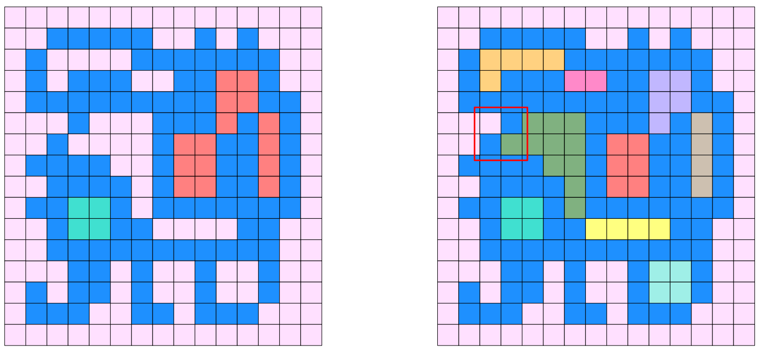


Figure 2 Example of a configuration (in blue) with 3 holes and 10 pseudo-holes. Left: Empty cells marked with the same color belong to the same hole. Right: Empty cells marked with the same color belong to the same pseudo-hole. The red square indicates one of the many critical pairs.

We call the *boundary* of a hole (respectively, pseudo-hole) the cycle of vertices and edges simultaneously incident to the hole (pseudo-hole) and the robot, and *boundary traversal* the cycle of hole (pseudo-hole) cells incident to its boundary.

A *critical pair* is a pair of vertex-adjacent robot modules such that no module exists that is edge adjacent to both. They can be found in pseudo-holes that are not holes. Figure 2 shows an example.

1.2 Problem Statement and Results

Given any two configurations C and C' with n modules each, we present an efficient algorithm that reconfigures C into C' in-place, without disconnecting the robot at any moment throughout the reconfiguration. The algorithm is centralized and sequential, i.e., it slides one module at a time. It is efficient in the sense that the overall slide moves performed along the reconfiguration is $\Theta(n^2)$, which is optimal if constant force and velocity are assumed [2]. Furthermore, if B and B' respectively are the 1-cell offset of the bounding boxes of C and C', and B and B' share their left-bottom corner cell, then space used throughout the reconfiguration is enclosed in $B \cup B'$.

Furthermore, we provide an on-line simulator of our algorithm, together with a first analysis of its performance in practice on a variety of configurations of different sizes.

1.3 Related Work

The sliding-square/cube model was introduced in [4] as a geometric abstraction for a variety of modular robots, such as Metamorphic [5], Vertical [8], Molecube [14], M-TRAN [10], EM-cube [3], or Smart [11]. Since then, several algorithms have been proposed to solve different (but somehow related) problems such as locomotion [4, 7], self-repair [9], and reconfiguration [6, 1].

In particular, [6] proposes the first algorithm to reconfigure between any two edge-connected shapes of sliding squares in the plane with the same number of modules. The algorithm is sequential and reconfiguration is done through an intermediate shape: a strip grown from one extremal module. As a consequence, the overall reconfiguration takes place in the disjoint union of the 1-offset of the bounding boxes of the initial and final configurations.

The algorithm we present builds on the ideas from [6], but differs from it in that it reconfigures in-place, i.e., within the 1-offset of the union of the two bounding boxes. This adds further difficulties, as in our case the final destination of a module may be located in a hole or a pseudo-hole, and not only in the outer boundary as in [6]. We further elaborate on this issue in the following section.

2 Algorithm overview

Given two configurations C and C', our strategy to reconfigure C into C' consists of three steps, illustrated in Figure 3.

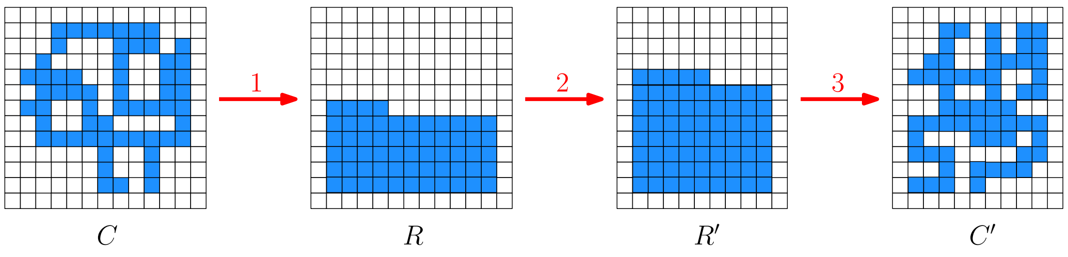


Figure 3 Reconfiguration steps.

1. First, we *flood* the bounding box of our initial configuration C. This process consists of sequentially finding modules that can slide without disconnecting the configuration, and sending them along a boundary traversal to fill the bounding box of C by rows from bottom to top, filling each row from left to right. The result is a rectangle, except for its topmost row, that may be incomplete. We call this shape R.
2. Then, we reconfigure R into R', which is the analogous almost-rectangular shape corresponding to C'.
3. Finally, we reconfigure R' into C'.

The algorithm for step 2 is straightforward, and that of step 3 consists of reversing the procedure of step 1. Therefore, in the remaining of this paper we concentrate on step 1.

As already mentioned, when flooding, the final destination of a module may be located in a hole or a pseudo-hole, and not only in the outer boundary as in [6]. In other words, while in [6] the algorithm consists in finding a feasible sequence of moves (of possibly several modules) that ends up freeing a module in the outer boundary, our algorithm needs to produce sequences that first go outwards from a pseudo-hole towards the outer boundary and then may have to go inwards to a different pseudo-hole in order to fill the goal position. Figure 4 illustrates this process.

We solve this issue using a hierarchy of pseudo-holes that is built by traversing all the boundaries of the configuration in a preprocessing step. Starting at the outer one, the boundary traversal detects all critical pairs. Each critical pair is a gate for a pseudo-hole that is a descendant in the hierarchy. Gates between pseudo-holes contained in holes different than the outer one are considered siblings in the hierarchy. Figure 5 illustrates this.

Once the hierarchy has been built, the algorithm looks for a module that is able to directly move to the goal cell, i.e., a module that is not a cut vertex of the edge-adjacency graph of the configuration, and is adjacent to the same pseudo-hole boundary as the next cell position to be filled. If such module does not exist, the algorithm searches through the pseudo-holes hierarchy for a module that can be moved, and sends it to connect a critical pair in the path to the goal pseudo-hole. This creates a cycle in the adjacency graph of C enclosing the goal cell, which, at its turn, allows finding a new module that can be moved inside the cycle. The process continues until a movable module can be found in the goal pseudo-hole. This, at last,

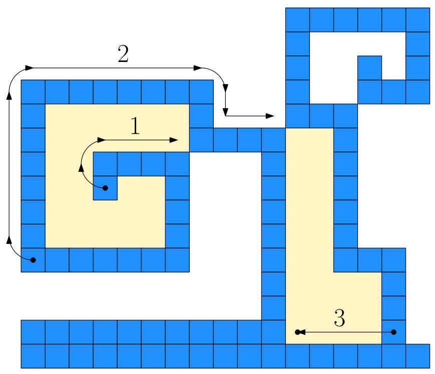


Figure 4 Filling the next free position in the second row. Notice how the sequence of moves starts in a pseudo-hole, continues in the outer boundary, and ends back in a (different) pseudo-hole.

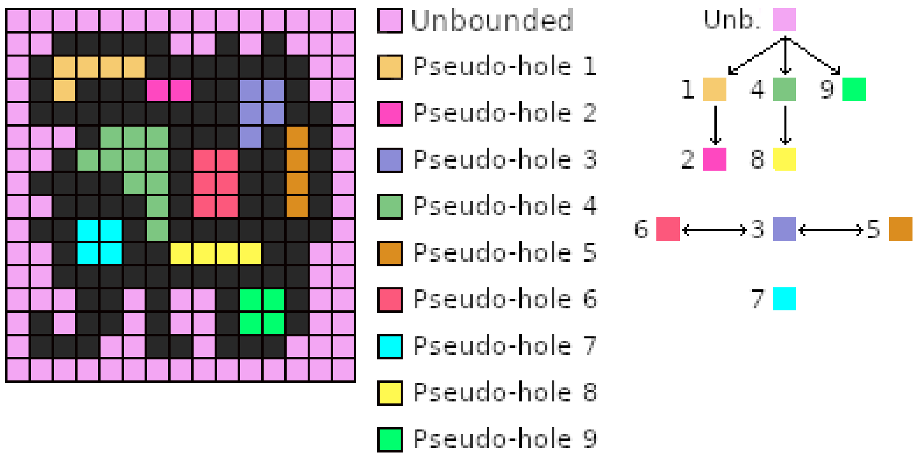


Figure 5 A Hierarchy of pseudo-holes.

fills the goal cell. Naturally, every time a modules leaves a position in the configuration, and every time it stops either because it reaches the goal cell or because it closes a critical pair, the hierarchy is updated accordingly.

This procedure is repeated until the configuration has been completely flooded.

3 Correctness and complexity

We denote by G the edge-adjacency graph of the configuration C, and by T the cactus graph associated WITH G, i.e., the tree of maximal cycles of G. We start by proving two lemmata.

▶ **Lemma 3.1.** *As long as not all the modules are in their final destination in R, there always exists a module that can slide without disconnecting the configuration.*

Proof. Since T is a tree, it must have a leaf. If such a leaf is a module, it must be movable since it is a leaf in G. Otherwise, if the leaf is a cycle, it must contain a module that is not a cut vertex of T. Such a module is movable since it cannot be a cut vertex in G. ◀

▶ **Lemma 3.2.** *If a movable module cannot reach the goal cell, g, it can connect a critical pair reducing the size of the cycle containing g.*

Proof. Let h_g be the pseudo-hole containing g, and let h_m be the pseudo-hole containing a movable module m whose existence is guaranteed by Lemma 3.1. A path must exist in the pseudo-hole hierarchy connecting h_g to h_m, since every pseudo-hole must belong to a hole, each hole is bounded by a cycle in T, and each cycle in T must contain a module that is not a cut vertex. ◀

▶ **Theorem 3.3.** *Let C be any configuration with n modules and R a configuration with the same number of modules filling the bounding box of C from bottom to top, left to right. Our algorithm reconfigures C into R using $\Theta(n^2)$ sliding moves.*

Proof. Each goal cell is filled after $O(n)$ slide moves. This is evident when the moving module reaches its destination without having to stop at a critical pair, since any boundary traversal has size $O(n)$. When it does stop, it can be proved that the overall number of slide moves performed by the sequence of moving modules before filling the empty goal position is $O(n)$. Details are omitted due to space restrictions. Therefore, the entire flooding takes $O(n^2)$ slide moves. It is worth noticing that the second step of the algorithm (reconfiguring between R and R') can be trivially done with $O(n^2)$ slide moves. Therefore, the entire reconfiguration uses $O(n^2)$ slide moves, which is optimal within this context [2]. ◀

4 Simulator and practical experiments

In addition to the theoretical results, we have also implemented an on-line simulator of our reconfiguration algorithm, see [12]. The simulator allows the user to define the initial and the final configurations in a 2-dimensional grid both by uploading a predefined file or by direct interaction. Then, it visualizes all the slide moves that the robot would be undertaking, step by step from the initial configuration C to the final one, C'. The reconfiguration can be stopped at any moment and backtracking is also allowed.

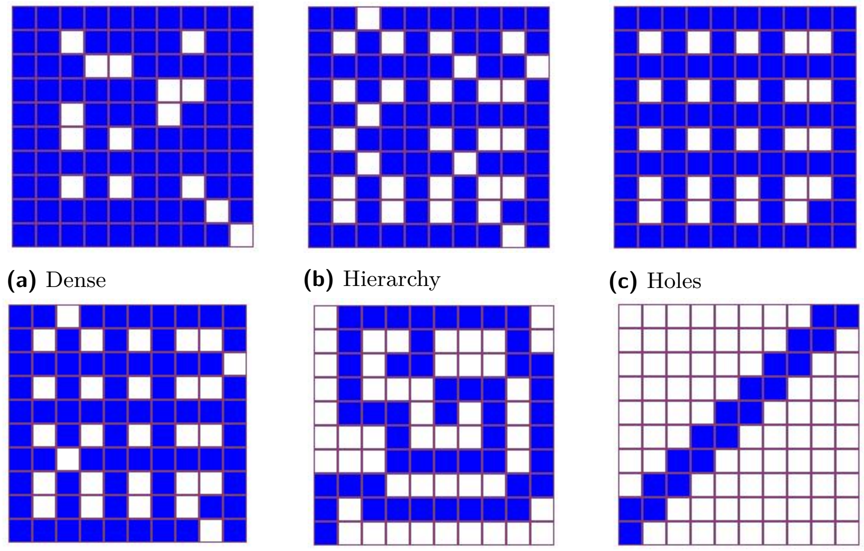
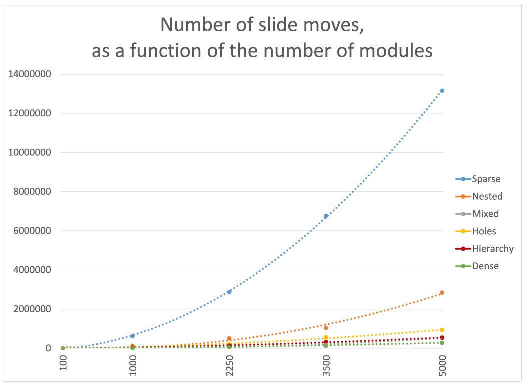
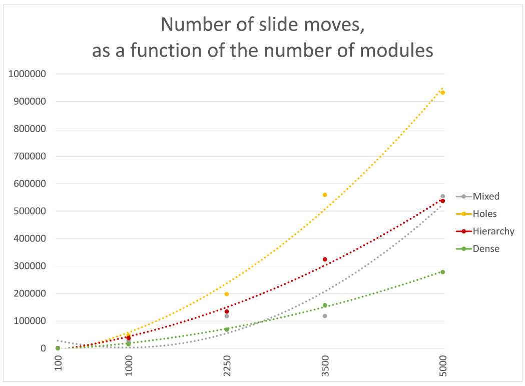

(a) Dense **(b)** Hierarchy **(c)** Holes **(d)** Mixed **(e)** Nested **(f)** Sparse

Figure 6 Examples of the configuration shapes used in our experiments.

We have used our simulator to test our algorithm on a variety of configurations, different in shape and size. The chosen shapes have been the following: Dense (configurations whose bounding box is almost completely filled with modules), Hierarchy (configurations with several and large hierarchies of pseudo-holes), Holes (configurations without hierarchies but with multiple holes), Mixed (configurations combining large hierarchies of pseudo-holes with multiple holes), Nested (configurations requiring to connect critical pairs in order to reconfigure), Sparse (configurations with maximal ratio between the size of their bounding box and their number of modules). Figure 6 shows examples of these shape types.

The results of the experiments show that the sparser a configuration is, more slide moves its reconfiguration requires. This is a consequence of the fact that sparsity usually obliges the modules to slide over a great number of other modules in order to reach the intermediate rectangular shape. Denser configurations require a smaller number of moves because many modules are already located in their final grid positions. Figure 7 shows the results of the experiments we run.





Number of slide moves,
as a function of the number of modules

		Number of modules				
		100	**1000**	**2250**	**3500**	**5000**
Shape	**Dense**	422	15554	68765	157662	278332
	Hierarchy	1415	37582	134395	324502	537519
	Holes	433	46908	197741	559821	931817
	Mixed	808	27491	117543	118319	554274
	Nested	3061	122339	505382	1034488	2843997
	Sparse	7806	621082	2877678	6760828	13156460

Figure 7 Number of slide moves used to reconfigure the different configuration types into their corresponding rectangles.

Notice that the number of slide moves indicated in the table and charts corresponds to reconfiguring each initial configuration C into its corresponding rectangle R. The number of slide moves corresponding to the overall reconfiguration of an initial shape C into a final shape C' can be obtained by adding up the corresponding values from the table together with the number of steps required to reconfigure R into R', which is straightforward to compute.

5 Conclusions and open problems

We have presented an algorithm that reconfigures between any two edge-connected configurations of n sliding squares within their bounding boxes by means of $\Theta(n^2)$ slide moves, and we have made available an on-line simulator. We believe that extending our strategy to the hexagonal and triangular grids should be straightforward.

Our algorithm is intrinsically sequential. An interesting open problem is to obtain a new strategy that allows to move several modules in parallel, giving rise to a faster reconfiguration, and to distribute it in order to obtain a more scalable algorithm. Extending the results to the 3-dimensional setting is also desirable.

6 Acknowledgments

We thank Diane Souvaine and Matias Korman for our preliminary discussions on a similar problem.

References

1 Z. Abel and S.D. Kominers. Universal reconfiguration of (hyper-)cubic robots. *CoRR*, abs/0802.3414, 2008. URL: `http://arxiv.org/abs/0802.3414`, `arXiv:0802.3414`.

2 G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, V. Pinciu, S. Ramaswami, V. Sacristán, and S. Wuhrer. Efficient constant-velocity reconfiguration of crystalline robots. *Robotica*, 29(1):59–71, 2011.

3 B.K. An. EM-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3155, 2008.

4 Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, page 809–816, 2002.

5 G.S. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 449–455, 1994.

6 A. Dumitrescu and J. Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006.

7 R. Fitch and Z. Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *The International Journal of Robotics Research*, 27(3–4):331–343, 2008.

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. IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 2858–2863, 1998.

9 K. Kotay and D. Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2362–2369, 2004.

10 H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *International Journal of Robotics Research*, 27(3-4):373–386, 2008.

11 S. Mobes, G.J. Laurent, C. Clevy, N. Le Fort-Piat, B. Piranda, and J. Bourgeois. Toward a 2D modular and self-reconfigurable robot for conveying microparts. In *Proc. Second Workshop on Design, Control and Software Implementation for Distributed MEMS (dMEMS)*, pages 7–13, 2012.

12 J. Moreno. Reconfiguring sliding squares almost in-place. https://dccg.upc.edu/people/vera/teaching/tfm-tfg/flooding/.

13 M. Yim, P. White, M. Park, and J. Sastra. Modular self-reconfigurable robots. In R.A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. Springer New York, 2009.

14 V. Zykov, A. Chan, and H. Lipson. Molecubes: An open-source modular robotic kit. In *IROS-2007 Self-Reconfigurable Robotics Workshop*, 2007.