An Interactive Framework for Reconfiguration in the Sliding Square Model

Willem Sonke
TU Eindhoven, The Netherlands

Jules Wulms
TU Wien, Austria

Abstract

We describe SquareSlider, a software framework for visualizing reconfiguration algorithms of modular robots in the sliding square model. In this model, a robot consists of a configuration of squares in a rectangular grid, which can reconfigure through a fixed set of possible moves. SquareSlider is a web-based tool that implements an easy-to-use interface allowing the user to build a configuration, run a reconfiguration algorithm on it, and examine the results.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Modular robots, Implementation, Visualization

Digital Object Identifier 10.4230/LIPIcs.SoCG.2022.70

Category Media Exposition

Supplementary Material
Software (Web Application): `https://alga.win.tue.nl/software/squareslider/`
archived at `swh:1:dir:4970151973ae26eeaac474506f7e0fa400e414db`

Acknowledgements We thank Bettina Speckmann for her useful comments on a draft of this paper.

1 Introduction

A popular research topic in robotics and computer science is the development of modular robots [9]. These typically consist of homogeneous building blocks called *modules*, connected in such a way that the robot is able to move modules relative to each other. This way, the robot can transform by *reconfiguring* its modules. Such a reconfiguration requires careful motion planning and efficient algorithms to be viable in practice. To enable the systematic study of reconfiguration algorithms, many models of modular robots have been proposed, such as the *sliding cube model* [3, 5, 6, 7] and the *pivoting cube model* [1, 2, 4, 8].

Here, we focus on reconfiguration in the sliding cube model in two dimensions (the "*sliding square model*"). In this model, the elementary building blocks are square modules that live in the rectangular unit grid. The modules can perform two types of moves: *slides* and *convex transitions* (illustrated in Figure 1). The source and target configurations are assumed to be face-connected, and at any time during the reconfiguration, the configuration also has to stay face-connected.

In this paper we describe a software tool called *SquareSlider* which provides a framework to visualize and interact with reconfigurations. We initially created SquareSlider to support the development of the reconfiguration algorithm Gather&Compact [3], but the framework can be useful for other algorithms as well. To this end, we designed SquareSlider to be modular: algorithm implementations are separate from the core. This way, although only Gather&Compact has currently been implemented, other algorithms can be plugged in easily.

We implemented SquareSlider as a web-based tool, thus ensuring that the visualization is easily accessible, without the additional hurdle of having to compile and run a separate program. SquareSlider is available at `https://alga.win.tue.nl/software/squareslider/`.

Figure 1 The two types of moves in the sliding square model: **(a)** slide, **(b)** convex transition.

For the implementation, we used TypeScript,[1] a typed variant of JavaScript that compiles down to JavaScript. For the graphics, we used WebGL via the PIXI.js[2] project. The GUI elements used (toolbars and buttons) are custom-built. In the remainder of this paper, we will describe in detail the functionality offered by our framework.

2 Functionality

SquareSlider's GUI consists of a large canvas that displays the configuration, and two toolbars: one on the top and one on the bottom (see Figure 2). The functionality of SquareSlider can be divided into four categories: visualization, interaction, animation, and framework utilities. We will highlight the most interesting aspects of each category.

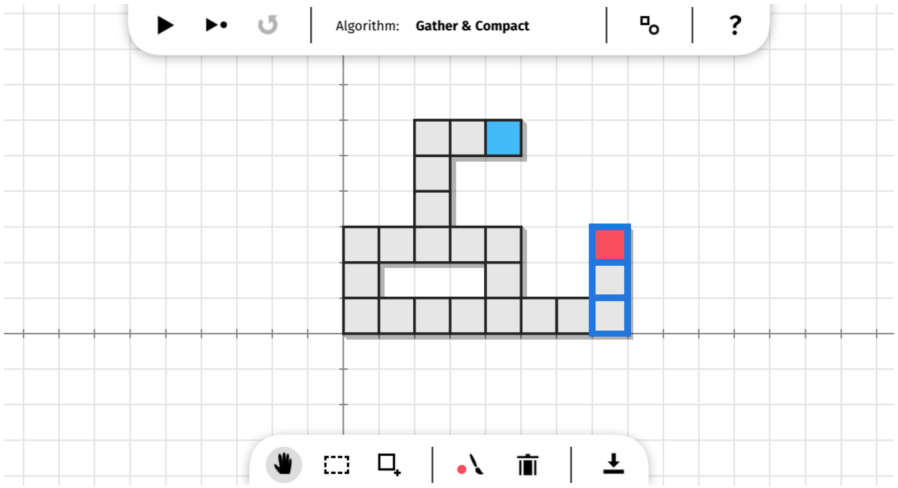

Figure 2 The GUI of SquareSlider after drawing a configuration and coloring two arbitrary squares. The three squares on the right have been selected, as indicated by the blue outlines.

Visualization. By default, a configuration is visualized as gray squares in a grid, as shown in Figure 3a. The grid has an x- and y-axis defining a coordinate system, where every grid cell is indexed by its bottom-left corner. The coordinates are especially useful for interaction and animation, as will be explained below.

Many reconfiguration algorithms use information about the connectivity of the configuration, such as partitioning the squares into components, depending on their k-connectivity. We therefore augment the visualization of the squares with marks indicating this information. Currently, SquareSlider can capture the type of connectivity used by Gather&Compact [3] (see Figure 3b). Firstly, a square has a filled (blue or red) mark if it can perform moves without disconnecting the configuration, while a non-filled mark indicates that moving the square

[1] `https://www.typescriptlang.org/`
[2] `https://pixijs.com/`

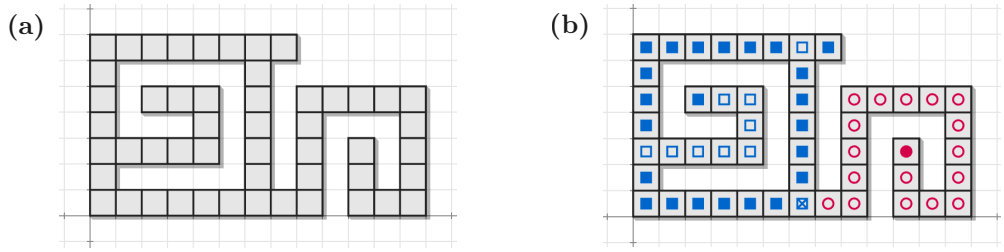

Figure 3 **(a)** A configuration visualized in our framework. **(b)** The same configuration, with connectivity marks added (see the main text for an explanation of the marks).

disconnects the configuration (i.e., this square forms a cut vertex in the adjacency graph of the configuration). Secondly, the color and shape of the marks indicates a partitioning of the configuration into (roughly) 2-connected components. Blue square marks indicate squares in a *chunk*, a variant of a 2-connected component introduced in [3]. Specifically, a chunk is defined by an inclusion-maximal cycle C along with all its degree-1 neighbors and any squares inside C. Red circle marks indicate squares that are not part of a chunk. A special crossed mark shows where components connect to each other.

We use a variant of the classic algorithm to compute 2-connected components in graphs, which is based on DFS. This algorithm can easily be adapted for different notions of connectivity, as may be required by other reconfiguration algorithms.

Interaction. Input configurations can be provided to SquareSlider in two ways. Firstly, the editing tools in the bottom toolbar (see Figure 2) allow users to place and remove squares by clicking or dragging over grid cells. Clicking an empty grid cell adds a square in that cell, while clicking an existing square removes that square. Dragging over a series of cells adds or removes squares in all these cells, thus enabling quick editing of large configurations. Furthermore, users can color squares to see how a specific square or set of squares moves during reconfiguration. Secondly, users can export the configuration to a JSON string, which can later be loaded again. Such a JSON string can also be generated by an external tool.

Animation. Once a connected configuration is built, a reconfiguration can be performed. The top toolbar provides functionality to start an animation of the moves in such a reconfiguration.

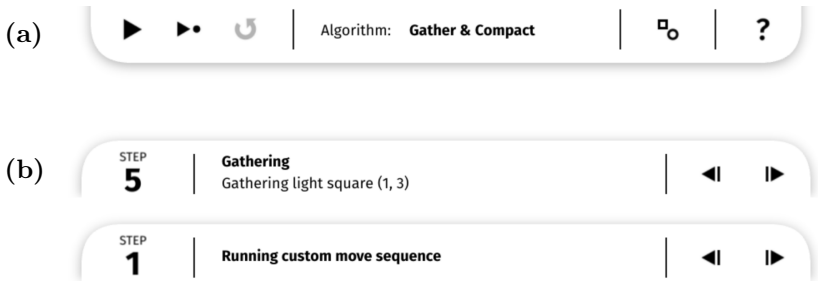

Figure 4 The toolbars for animating reconfigurations. **(a)** The top toolbar. The left two buttons allow the user to execute a reconfiguration algorithm, either by playing the whole sequence of moves, or just a single step. The circular arrow button resets the configuration to the state before reconfiguration. **(b)** The bottom toolbar (shown here for both Gather&Compact and a custom move sequence) showing status information about the currently running algorithm.

While the reconfiguration is running, the bottom toolbar is repurposed to show details on the performed moves (see Figure 4). Functionality is available to play the reconfiguration in one go (at various speeds), walk through the reconfiguration step by step, and to reset the configuration to the original state (after which the bottom toolbar returns to its original state, such that the configuration can be edited again).

Besides running an algorithm implementation, SquareSlider also has the option to manually input a JSON string containing a sequence of moves. This can be useful to interface with external tools. For example, the experiments in [3] used this functionality to interface with the original implementation of [7].

Framework utilities. SquareSlider provides a wide range of utility functions that can be used by algorithm implementations. For example, this includes functionality to check if a configuration is connected, to determine if a given move is valid, to enumerate all squares on the outer boundary of the configuration, and to compute a shortest sequence of moves to move a given square to a target location.

An algorithm implementation consists of a TypeScript class containing a function that produces moves that the algorithm wishes to perform. To ensure robustness, the core checks if moves performed by an algorithm are valid in the sliding square model. Any invalid moves halt the execution of the algorithm.

References

1 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

2 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Dylan H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.

3 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. *CoRR*, abs/2105.07997, 2021. arXiv:2105.07997.

4 Nora Ayanian, Paul J. White, Ádám Hálász, Mark Yim, and Vijay Kumar. Stochastic control for self-assembly of XBots. In *Proc. ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE)*, pages 1169–1176, 2008. doi:10.1115/DETC2008-49535.

5 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006. doi:10.1007/s00373-005-0640-1.

6 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. 2003 IEEE/RSJ International Conference on Intelligent Robots and System*, volume 3, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

7 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proc. 36th European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020.

8 Cynthia Sung, James Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *Proc. 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1933–1940, 2015. doi:10.1109/ICRA.2015.7139451.

9 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007. doi:10.1109/MRA.2007.339623.