Sliding Squares in Parallel*

Hugo A. Akitaya[1], Sándor P. Fekete[2], Peter Kramer[2], Saba Molaei[3], Christian Rieck[4], Frederick Stock[1], and Tobias Wallner[2]

1 **Miner School of Computer and Information Sciences, University of Massachusetts Lowell, MA, USA**
hugo_akitaya@uml.edu, frederick_stock@student.uml.edu

2 **Department of Computer Science, TU Braunschweig, Braunschweig, Germany**
s.fekete@tu-bs.de, {kramer, wallner}@ibr.cs.tu-bs.de

3 **Sharif University of Technology, Tehran, Iran**
molaeisaba1@gmail.com

4 **Department of Discrete Mathematics, University of Kassel, Kassel, Germany**
christian.rieck@mathematik.uni-kassel.de

Abstract

We consider algorithmic problems motivated by modular robotic reconfiguration, for which we are given n unlabeled square-shaped modules in a starting configuration and need to find a schedule of sliding moves to transform it into a desired goal configuration, maintaining connectivity at all times.

Recent work has aimed at minimizing the total number of moves, resulting in fully sequential schedules that can perform reconfigurations in $\mathcal{O}(n^2)$ moves, or $\mathcal{O}(nP)$ for an arrangement with bounding box of perimeter P. We provide results in the sliding square model that exploit parallel robot motion, resulting in an optimal speedup to achieve reconfiguration in worst-case optimal makespan of $\mathcal{O}(P)$. We also show a tight bound on the complexity of the problem by showing that even deciding the possibility of reconfiguration within makespan 1 is NP-complete.

Related Version 10.48550/arXiv.2412.05523

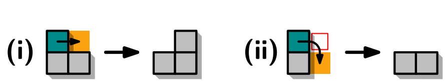
1 Introduction

Reconfiguring an arrangement of objects is a fundamental problem in both theory and practice, often in a setting with strong geometric flavor. A typical task arises from relocating a (potentially large) collection of agents from a given start into a desired goal configuration in an efficient manner, while avoiding collisions between objects or violating other constraints, such as maintaining connectivity of the overall arrangement.

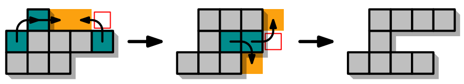
In recent years, the problem of modular robot reconfiguration [7, 18, 19] has enjoyed particular attention [1, 2, 3, 4, 6, 15] in the context of Computational Geometry: In the *sliding square model* introduced by Fitch, Butler, and Rus [13], a given start configuration of n modules, each occupying a square grid cell, must be transformed by a sequence of atomic, sequential moves (shown in Figure 1(a)) into a target arrangement, without losing connectivity of the underlying grid graph. Aiming at minimizing the total number of moves, the mentioned previous work has resulted in considerable progress, recently establishing universal configuration in $\mathcal{O}(nP)$ for a 2-dimensional arrangement of n modules with bounding box perimeter size P [4], and $\mathcal{O}(n^2)$ in three dimensions [1, 15]. This work on sequential reconfiguration differ from practical settings, in which modules can exploit parallel motion to achieve much faster reconfiguration times—which is also a more challenging objective, as it requires coordinating the overall motion plan to maintain connectivity and avoid collisions.

* This work was partially supported by the German Research Foundation (DFG), project "Space Ants", FE 407/22-1, by NSF grant CCF-2348067, and by DFG grant 522790373.

Our work focuses on parallel reconfiguration, allowing modules to perform moves simultaneously with the objective of minimizing the total time until completion, the *makespan*. For a detailed overview of related literature and recent results, we refer to the full version [5].



(a) Two types of move can be performed by individual modules: **(i)** Slides and **(ii)** convex transitions.



(b) Some moves can be performed in parallel transformations, as shown here.

Figure 1 Our model allows for two types of moves to be chained into collision-free transformations. In this paper, we show the symmetric difference of a transformation using turquoise and yellow.

Our contributions. We provide the following tight results for *parallel* reconfiguration in the sliding square model. For technical details, we refer to the full version of our paper [5].

1. We prove that the *unlabeled* version of parallel reconfiguration is NP-complete, even when trying to decide the existence of a schedule with the smallest possible makespan of 1.
2. We give a weakly in-place algorithm that achieves makespan $\mathcal{O}(P)$, where P is the perimeter of the union of the bounding boxes of start and target configurations.

Preliminaries. We study reconfiguration in the *parallel sliding square* model as follows. An *instance* $\mathcal{I}$ is composed of an initial configuration C_1 of n robotic *modules* that must be reconfigured into a target configuration C_2. In any configuration, each square module occupies a unique cell of the infinite integer grid. We navigate the grid using cardinal directions (north, south, east, west) and cell adjacency. To reconfigure C_1 into C_2, we employ schedules of atomic, parallel *transformations*. A transformation selects a subset of modules to move, each performing either a *slide* or a *convex transition* to travel into an adjacent cell, see Figure 1(a). Note that in our figures, the path denoting a convex transition is shown containing a circular arc for clarity. Such path would be accurate if the corners of modules were rounded. In our model, both take an identical (unit) duration to complete, which allows us to extend the existing notion of move counting in sequential models [4, 9, 16, 10] to transformation counting for parallel moves. The *makespan* of a reconfiguration schedule thus corresponds to the number of transformations.

During each transformation, a connectivity-preserving *backbone* must be maintained. To this end, we call a set M of modules in a configuration C *free* if $C \setminus M'$ is a valid configuration for any $M' \subseteq M$; a transformation $C_1 \to C_2$ is legal exactly if the moving modules are free in C_1 and C_2. Furthermore, a transformation is collision-free exactly if all modules can move along their designated path at a constant rate, without overlapping with any other module in the process. We identify three types of collisions that must be avoided, see Figure 2.

(i) Any two moves collide if their target cells are identical, or if they constitute a swap.
(ii) Two convex transitions collide if they share an intermediate cell (highlighted in red).
(iii) Two moves collide if their paths meet orthogonally at endpoints.

In the full version of our paper, we discuss these in depth and show that some collisions of type **(iii)** can be avoided with only a constant overhead in the number of transformations. Our collision model is less restrictive than some models of previous work on parallel reconfiguration [10, 14], which require the motion paths of parallel moves to be pairwise disjoint. On the other hand, it is more constrained than [11] which does not forbid collisions of type **(iii)**. (However, note that [11] does not impose the single backbone condition.)

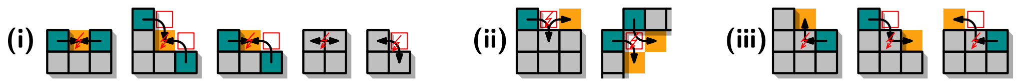


Figure 2 Examples of all collision types: Modules cannot share or swap cells, convex transitions cannot share an intermediate cell, and moves with connected paths cannot meet orthogonally.

Consider an instance $\mathcal{I} = (C_1, C_2)$. Throughout this paper, we denote the bounding boxes of C_1 and C_2 by B_1 and B_2 , respectively. As in existing literature, we assume that B_1 and B_2 share a south-west corner and call a schedule for $\mathcal{I}$ *in-place* exactly if no intermediate configuration exceeds the union $B_1 \cup B_2$ by more than one module [16, 4]. For technical reasons, we desire bounding box dimensions that are multiples of three, as well as a three-wide empty column at the eastern boundary. We define the *extended bounding box* B' from B by expanding it in each of the cardinal directions by at most three units so that the dimensions of B' are multiples of three (see Figure 5). We say that a schedule is *weakly in-place* if and only if intermediate configurations are restricted to the union of the bounding boxes extended by a constant amount, e.g., $B'_1 \cup B'_2$. We refer to the standard definition as *strictly in-place*.

2 Computational complexity

In this section, we sketch a reduction from PLANAR MONOTONE 3SAT [8] showing that PARALLEL SLIDING SQUARES is NP-complete. In particular, we prove the following theorem.

▶ **Theorem 2.1.** *Let $\mathcal{I}$ be an instance of PARALLEL SLIDING SQUARES. It is* NP*-complete to decide whether there exists a feasible schedule of makespan* 1 *for $\mathcal{I}$.*

An example of the overall construction is depicted in Figure 3. The involved *variable* and *clause gadgets* are marked in yellow and blue, respectively. While the gadgets mostly retain their shape in the target configuration (visualized as gray modules), a single module must be moved per variable gadget. There are two feasible chain moves for this, each representing either a positive or negative Boolean value assignment, as shown in Figure 3.

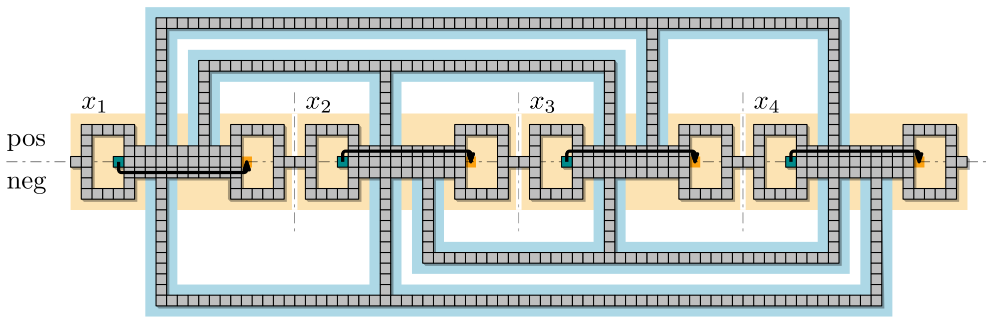


Figure 3 Our construction for $\varphi = (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$. The depicted transformation represents the satisfying assignment $\alpha(\varphi) = (\texttt{true}, \texttt{false}, \texttt{false}, \texttt{false})$.

Proof sketch. Due to the way in which both configurations are connected, the transformation depicted in Figure 3 disconnects each variable gadget from either all its positive or negative clauses. Thus, a satisfying assignment maps to a feasible schedule. As these chain moves are unique schedules of makespan 1, we also easily obtain the opposite direction. ◀

It is easy to solve our reduction instances in makespan 2, implying the following.

▶ **Corollary 2.2.** *Unless* P = NP*, there is no polynomial-time* $(1+\delta)$*-approximation algorithm for* Parallel Sliding Squares *with* $\delta \in [0, 1)$.

Our result highlights a complexity gap between the parallel sliding square model and closely related models for parallel transformation. In particular, Fekete et al. [11, 12] studied a related model in which makespan 2 is NP-hard to decide, but makespan 1 is in P.

3 A worst-case optimal algorithm

We now introduce a four-phase, polynomial-time algorithm for Parallel Sliding Squares. The high-level goal of our algorithm is to transform the entire configuration into *meta-modules*, small units of modules that cooperate to provide greater reconfiguration capability, which we then exploit. Unlike our work, the existence of such meta-modules is a common constraint on the input in related models [6, 11, 12, 14, 17] due to their high capability.

A meta-module in our work has a distinct *center cell* v surrounded by eight modules, such that all vertex-adjacent cells to v are full; the cell v itself may be empty or occupied.

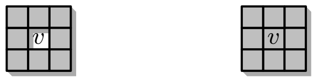


Figure 4 Meta-modules are either clean or solid, depending on whether the center cell is occupied.

Our approach computes reconfiguration schedules linear in the perimeter P_1 and P_2 of the bounding boxes B_1 and B_2 of C_1 and C_2, respectively. This is asymptotically worst-case optimal. Instances that induce a matching lower bound can easily be constructed, e.g., an instance asking for a large "L" to be rotated into "Γ" requires at least $\Omega(P_1 + P_2)$ moves.

▶ **Theorem 3.1.** *For any instance* $\mathcal{I}$ *of* Parallel Sliding Squares*, we can compute a feasible, weakly in-place schedule of* $\mathcal{O}(P_1 + P_2)$ *transformations in polynomial time.*

Our algorithm consists of four phases: In **Phase (I)**, we identify a subconfiguration used as a "backbone" (similar to [14]), and gather $\Theta(P_1)$ many modules around a piece of this backbone, thereby enhancing its connectivity (as in [4]). We use the flexibility of this piece to construct a sweep-line structure out of meta-modules in **Phase (II)**, that is then used in **Phase (III)** to efficiently compact and transform the remaining configuration into meta-modules forming an xy-monotone histogram, similar to [1]. In **Phase (IV)**, this histogram can then be transformed into any other such histogram with the same number of modules, effectively morphing between a "start histogram" and "target histogram", using similar techniques as in [11]. To reach our target configuration, we then simply apply **Phases (I-III)** in reverse. We refer to Figure 5 for a visualization of the different phases of the algorithm. Although several of our techniques are inspired by previous work, they differ substantially in our context of parallel reconfiguration and considerable changes were needed.

Phase (I): Gathering squares. We use an underlying connected substructure, referred to as a *skeleton*, from the initial configuration to guide the reconfiguration. This tree-like skeleton functions as a backbone around which we move modules toward a "root" module h, making a subtree of the skeleton "thick" (i.e., the neighborhood of every cell in this part of the skeleton is occupied). We call this thick subskeleton an *exoskeleton*, denoted by X_h. The exoskeleton has a higher connectivity than the skeleton, making it easier to transform into a sweep line.

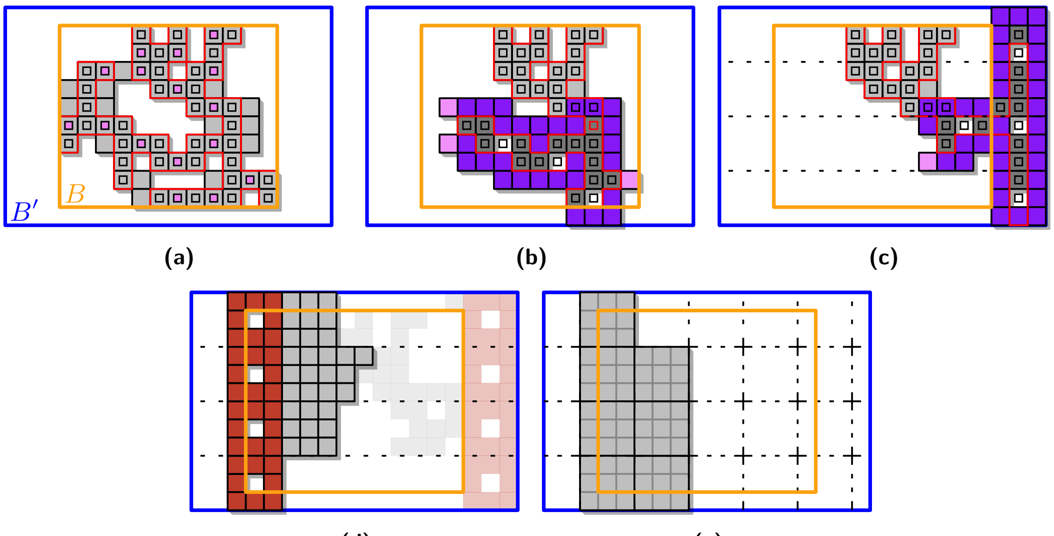


Figure 5 The high-level overview of our approach: We show (a) an initial configuration C and its skeleton (red border), (b) the gathered exoskeleton (in dark gray and purple) and (c) its resulting scaffold, (d) the sweep line (in red) in its initial and final state, and finally (e) the xy-monotone histogram of meta-modules.

A similar idea is also used by Hurtado et al. [14]. However, we achieve a stronger result in the classic sliding model via careful definition of the skeleton and movement. Upon completion of the gathering process, the exoskeleton X_h contains $\Theta(P)$ modules.

Phase (II): Scaffolding. Once the exoskeleton X_h is sufficiently large, we reconfigure it to be "T-shaped" and contain the east edge of the extended bounding box. Note that the interior (*core*) cells of the exoskeleton X_h are not necessarily all occupied; connectivity is instead maintained through their neighborhood. We proceed in three steps:

1. First "compact" X_h so that its core contains no empty cells.
2. Then choose an easternmost node c in the core of X_h and grow a horizontal path from c eastward (Figures 6(a) to 6(c)), until a 3×3 square is formed outside the bounding box.
3. Grow the path north until it reaches the corner of the bounding box, then grow a path south until it reaches the adjacent corner (Figures 6(d) to 6(f)).

Phase (III): Sweeping into a histogram. Once **Phase (II)** concludes, we are left with an intermediate configuration that contains a highly regular "scaffold" configuration with a wall of modules at the easternmost edge, as shown in Figure 6(f). Using only local operations, we transform this vertical section into meta-modules, forming our "sweep line" ℓ. We use local protocols to move the sweep line across the extended bounding box from east to west, consuming and pushing modules ahead of it, as shown in Figures 7(a) to 7(d). These consumed modules are pushed ahead of the sweep line, up to the western edge of the bounding box, at which point they start trailing behind in a "compacted" form, see Figures 7(a) to 7(c). This compacted form yields a number of histograms attached to ℓ, see Figure 7(f).

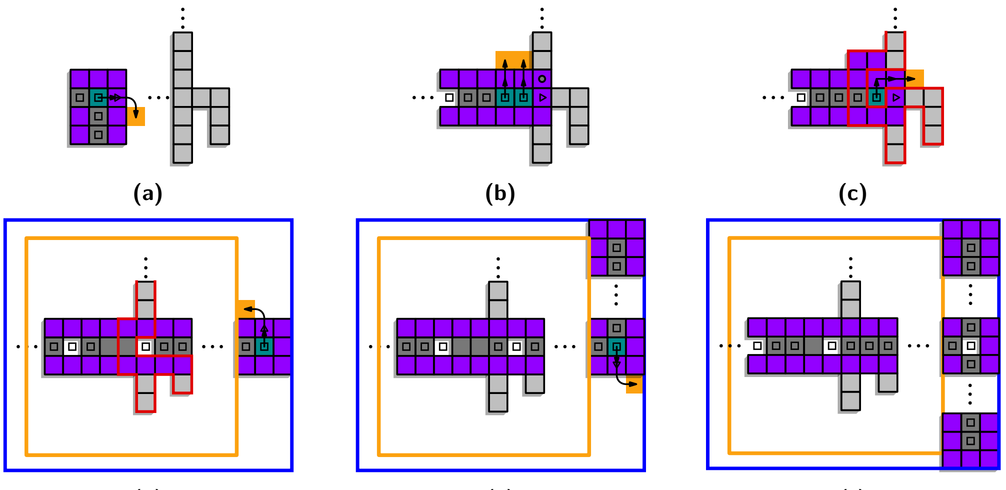


Figure 6 Building the scaffolding. The extended bounding box is shown in blue.

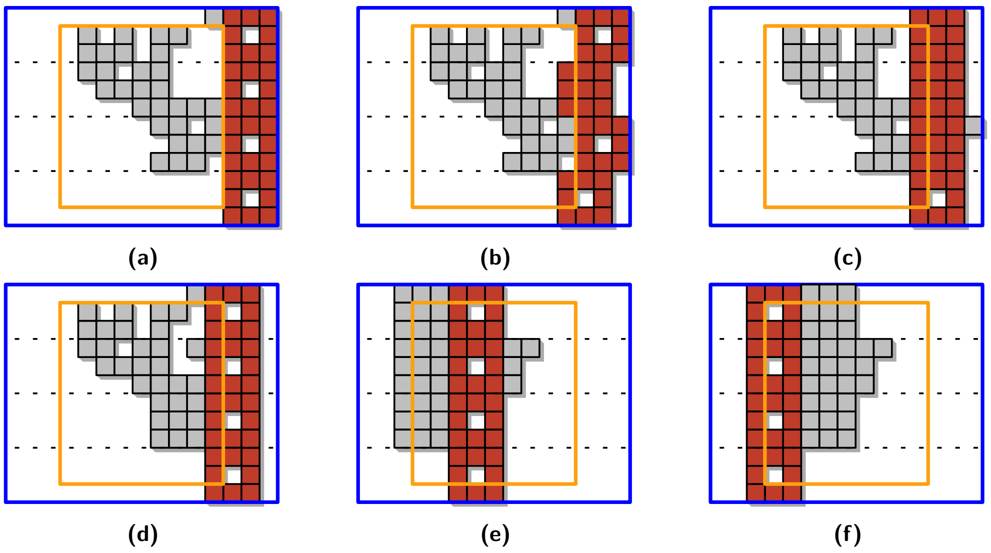


Figure 7 Moving the sweep line (red modules). The extended bounding box is shown in blue.

Phase (IV): Histograms of meta-modules. Once we complete the sweep, i.e., **Phase (III)**, we have a configuration that has a sweep line at the western boundary, with a compact configuration of modules east of the sweep line. This configuration resembles an x-monotone histogram of meta-modules, with at most a constant number of modules at each y-coordinate not belonging to a meta-module. We group the modules into meta-modules that form a histogram aligned with a regular grid, creating a *scaled histogram*, as visualized in Figure 8.

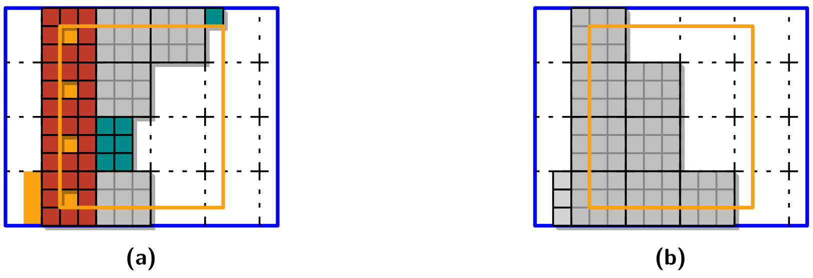


Figure 8 After the completing the sweep, we construct xy-monotone histogram of meta-modules.

Any two such histograms can be efficiently reconfigured into one another by a schedule that never places any module outside their union, i.e., a strictly in-place schedule. To realize this, we employ a simple sweep-line algorithm similar to [11, 12] that incrementally reduces the symmetric difference between the start and target histograms by mapping between excess occupied and unoccupied cells using two diagonal bisectors. Once we have obtained the scaled target histogram, we can begin applying the processes described in the above phases in reverse, eventually obtaining the target configuration.

Each phase takes $\mathcal{O}(P_1)$ transformations, so its reverse takes $\mathcal{O}(P_2)$. We therefore obtain an overall makespan of $\mathcal{O}(P_1 + P_2)$; a detailed analysis can be found in the full version.

4 Conclusions and future work

We have provided a number of new results for reconfiguration in the sliding squares model, making full use of parallel robot motion. While these outcomes are worst-case optimal, there are still a number of possible generalizations and extensions.

We can generalize our hardness results for the labeled setting to show that deciding whether makespan 2 is possible is NP-complete. Furthermore, we provide an efficient approach to decide if a schedule of makespan 1 exists. We can adapt our algorithmic results to this setting by "sorting" the modules in the xy-monotone configuration in $\mathcal{O}(P)$ transformations.

Previous work has progressed from two dimensions to three. Can our approach be extended to higher dimensions? We are optimistic that significant speedup can be achieved, but the intricacies of three-dimensional topology may require additional tools.

References

1 Zachary Abel, Hugo A. Akitaya, Scott Duke Kominers, Matias Korman, and Frederick Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves. In *Symposium on Computational Geometry (SoCG)*, pages 1:1–1:14, 2024. doi:10.4230/LIPIcs.SoCG.2024.1.

2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera

Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $\mathcal{O}(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

3 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 *Symposium on Computational Geometry (SoCG)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.

4 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. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 4:1–4:19, 2022. doi:10.4230/LIPIcs.SWAT.2022.4.

5 Hugo A. Akitaya, Sándor P. Fekete, Peter Kramer, Saba Molaei, Christian Rieck, Frederick Stock, and Tobias Wallner. Sliding squares in parallel, 2024. doi:10.48550/arXiv.2412.05523.

6 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry*, 42(6-7):652–663, 2009. doi:10.1016/J.COMGEO.2008.11.003.

7 Zack Butler and Daniela Rus. Distributed planning and control for modular robots with unit-compressible modules. *The International Journal of Robotics Research*, 22(9):699–715, 2003. doi:10.1177/02783649030229002.

8 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.

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

10 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *Transactions on Robotics*, 20(3):409–418, 2004. doi:10.1109/TRA.2004.824936.

11 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. *Autonomous Agents and Multi-Agent Systems*, 37(2):43, 2023. doi:10.1007/S10458-023-09626-5.

12 Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots. *Autonomous Agents and Multi-Agent Systems*, 38(2):39, 2024. doi:10.1007/S10458-024-09668-3.

13 Robert Fitch, Zack J. Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

14 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán Adinolfi. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015. doi:10.1007/S10514-015-9421-8.

15 Irina Kostitsyna, Tim Ophelders, Irene Parada, Tom Peters, Willem Sonke, and Bettina Speckmann. Optimal in-place compaction of sliding cubes. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 31:1–31:14, 2024. doi:10.4230/LIPICS.SWAT.2024.31.

16 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020. URL: https://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_32.pdf.

17 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. `doi:10.1007/S10514-021-09977-6`.

18 Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10:107–124, 2001. `doi:10.1109/MRA.2007.339623`.

19 Serguei Vassilvitskii, Mark Yim, and John Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *International Conference on Robotics and Automation (ICRA)*, pages 117–122, 2002. `doi:10.1109/ROBOT.2002.1013348`.