A Universal In-Place Reconfiguration Algorithm for Sliding Cube-Shaped Robots in a Quadratic Number of Moves

Zachary Abel
Massachusetts Institute of Technology, Cambridge, MA, USA

Hugo A. Akitaya
University of Massachusetts Lowell, MA, USA

Scott Duke Kominers
Harvard University, Cambridge, MA, USA
a16z crypto, New York, NY, USA

Matias Korman
Siemens Electronic Design Automation, Wilsonville, OR, USA

Frederick Stock
University of Massachusetts Lowell, MA, USA

Abstract

In the modular robot reconfiguration problem, we are given n cube-shaped *modules* (or *robots*) as well as two *configurations*, i.e., placements of the n modules so that their union is face-connected. The goal is to find a sequence of moves that reconfigures the modules from one configuration to the other using "sliding moves," in which a module slides over the face or edge of a neighboring module, maintaining connectivity of the configuration at all times.

For many years it has been known that certain module configurations in this model require at least $\Omega(n^2)$ moves to reconfigure between them. In this paper, we introduce the first universal reconfiguration algorithm – i.e., we show that any n-module configuration can reconfigure itself into any specified n-module configuration using just sliding moves. Our algorithm achieves reconfiguration in $O(n^2)$ moves, making it asymptotically tight. We also present a variation that reconfigures *in-place*, it ensures that throughout the reconfiguration process, all modules, except for one, will be contained in the union of the bounding boxes of the start and end configuration.

2012 ACM Subject Classification Theory of computation → Computational geometry

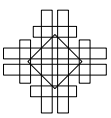
Keywords and phrases modular reconfigurable robots, sliding cube model, reconfiguration

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.1

Related Version This paper extends and subsumes an earlier preprint by Abel and Kominers [1].
Full Version: `https://arxiv.org/abs/0802.3414`

Funding *Scott Duke Kominers*: Part of this work was conducted during the Simons Laufer Mathematical Sciences Institute Fall 2023 program on the Mathematics and Computer Science of Market and Mechanism Design, which was supported by the National Science Foundation under Grant No. DMS-1928930 and by the Alfred P. Sloan Foundation under grant G-2021-16778.

Acknowledgements The authors would like to thank Maarten Löffler and for his contributions during early discussions as well as the authors of [11] and the anonymous reviewers for their valuable comments. Finally, we would like to thank Kevin Li and Colton Wolk for implementing preliminary versions of the algorithms proposed in this paper.



1 Introduction

A *modular self-reconfigurable robotic system* is a set of robotic units (called *modules*) that can communicate, attach, detach and move relative to each other; various models of such robots have received considerable attention by the computational geometry community [1, 6, 7, 9, 8, 2, 14]. When analyzing a model of modular robots, the typical goal is to find a *universal reconfiguration algorithm* defined as follows: A *(reconfigurable robot) configuration* is an arrangement of modules in space that is required to be connected. A *move* is a local rearrangement involving one module that transforms one configuration into another. During a move, the set of stationary modules is also required to be connected (known as the *single backbone condition* [9]).

We say that reconfiguration is *universal* if given any two configurations s and t, there is always a series of moves that reconfigures s to t; an algorithm that computes this sequence of moves is a *universal reconfiguration algorithm*. For many models of reconfigurable robots, universal reconfiguration is impossible [2, 14], and furthermore, it is often NP or PSPACE-complete to determine if one can even reconfigure between two configurations [3].

We consider the *sliding (hyper-)cube model*, in which each module is a (hyper-)cube, and a configuration comprises a placement of the cubes into lattice-aligned positions so that the interior of their union is connected. Two (d-dimensional hyper-)cubes are *adjacent* if they share a face (i.e., a $(d-1)$-dimensional facet). A module can slide along a face of an adjacent module, either moving to be adjacent to a new module or rotating around a corner of a module (see Figure 1). The *free-space requirement* for a move is the set of lattice positions that are required to be empty for a move to be collision-free.

Figure 1 (Left) Slide Move and (right) Rotation Move.

This sliding cube model has attractive properties relative to other popular reconfigurable robot systems. In the *pivoting model*, robots rotate around a shared edge instead of sliding on a face [14]; this model requires more free space, making reconfiguration more difficult or only possible in limited cases [2]. In the *crystalline model* [6], robots move via expansions and contractions, and universal reconfiguration requires $2 \times 2 \times 2$ *meta-modules*, small sub-arrangements of modules which are treated as the atomic units. Furthermore, $2 \times 2 \times 2$ crystalline meta-modules can simulate sliding moves and, thus, any reconfiguration algorithm for the sliding cube model can also be used for the crystalline model [5].[1]

The problem of sliding cube reconfiguration is fairly well understood in two dimensions: Dumitrescu and Pach [8] were the first to show a universal reconfiguration algorithm for n modular sliding 2-cubes, using $O(n^2)$ moves. Moreno and Sacristán [13] adapted their result to run *in-place*, i.e., at any time only one robot is outside the union of the bounding boxes of the start and target configurations. Recently, Akitaya et al. [4] showed that finding the shortest reconfiguration (in terms of number of moves) is NP-hard in 2D. They also improved on Moreno and Sacristán's algorithm by maintaining in-place property while simultaneously making the algorithm input-sensitive. Effectively, they reduced the number of moves to $O(\overline{P}n)$ moves, where $\overline{P}$ is the maximum perimeter of the two bounding boxes.

[1] The analogous statement is true for higher-dimensional systems.

For three (and higher) dimensions, sliding cube configuration has also received significant attention, yet not so much is known. The sliding cube model was first described by Fitch, Butler, and Rus [10]. They proposed a simple universal reconfiguration strategy for a configuration C of n modules: find a module $\mathbf{m}$ on the outer boundary of C that will not break connectivity if removed, move $\mathbf{m}$ on top of a fixed extreme position on the boundary, and repeat this operation $n-2$ more times. The Fitch, Butler, and Rus approach would lead to a reconfiguration algorithm that requires $O(n^2)$ moves. It is known that $\Omega(n^2)$ moves are sometimes necessary for reconfiguration in 3D (explicit construction shown in [12]), which would make their algorithm optimal.

Critically, Fitch, Butler, and Rus [10] rely on the continuous existence of the desired module $\mathbf{m}$. However, Miltzow et al. [12] recently presented a configuration where no such module exists. In other words, there are configurations of sliding cubes where no module on the outer boundary of C can move without breaking connectivity – implying that a more complex approach is necessary.

In 2008, Abel and Kominers announced a universal reconfiguration algorithm for dimensions 3 and higher in an arXiv preprint [1]. In essence, their algorithm requires $O(n^2)$ moves to adaptively ensure that the condition of Fitch, Butler, and Rus [10] is satisfied, which leads to an overall $O(n^3)$ algorithm. Their result was never formally published and unfortunately, the analysis has some minor flaws.[2] Parallel to our work, another group of researchers announced a universal in-place reconfiguration algorithm for three and higher dimensions [11]. Their algorithm is input-sensitive, and the number of moves is bounded by the overall sum of coordinates of modules in both configurations (values that range from $\Omega(n^{4/3})$ to $O(n^2)$).

Contribution

After we present the key definitions and model framework in Section 2, Section 3 introduces topological properties that are foundations of the algorithms presented in this paper. In Section 4, we show that a slight modification of the Abel–Kominers [1] algorithm achieves universal reconfiguration for the sliding (hyper-)cubes model. (In Section 4.1 we identify and fix a small mistake in the Abel–Kominers [1] manuscript.) Moreover, by improving the analysis (and with our minor corrections to the algorithm), we can reduce the required moves to $O(n^2)$, making the algorithm optimal.

In Section 5, we modify the algorithm further to obtain a new in-place algorithm for sliding cube reconfiguration; by *in-place* we mean during reconfiguration, moving modules stay within $O(1)$ distance of the union of the bounding boxes of the start and target configurations, as in [13]. In addition, our algorithm is input-sensitive, similar to Akitaya et al.'s result [4]. A natural 3-dimensional extension of their $O(\overline{P}n)$ algorithm would use $O(Vn)$ moves (V being the volume of the configuration) but our algorithm requires fewer – the bound is closer to $O(\overline{P}n)$ than $O(Vn)$; a formal statement is in Section 5.

2 Preliminaries: The Sliding Cube Model

Let M be a set of n distinct d-dimensional hypercube modules. A *labeled configuration* of M is an injective function from M to the set of unit cells in the d-dimensional hyper-cube lattice. (Here, by a *cell*, we mean an axis-aligned unit cube with vertices on the integer grid.) The image of a labeled configuration in the hypercubic lattice is called an *unlabeled configuration*.

[2] Historical note: This document subsumes and further extends the results presented in the Abel–Kominers preprint [1]. We acknowledge that a lot of time has passed since [1] was initially posted online, so in the remainder of this document, we treat it as a separate document.

For an unlabeled configuration C, we say that a lattice cell is *occupied* if it is in C; the cell is *empty* otherwise. We sometimes abuse terminology by referring to the occupied cells of C as simply the "cells of C." We refer to the $(d-1)$- and $(d-2)$-dimensional facets a lattice cell as *faces* and *edges*, respectively. Two cells are *adjacent* if they share a face.

Let G_C be the graph whose vertices are the cells of C with edges connecting pairs of cells that share one face. We say that C is connected if G_C is connected. Note that a connected configuration is a polyhypercube (a generalization of a polyomino to dimension d). Although we focus on unlabeled configurations, for clarity we may refer to them by way of their labeled counterparts, using language such as "move module $\mathbf{m}$ from cell a to cell b." We also sometimes abuse notation by referring to a cell by the module that occupies it in a configuration, for example, saying that two modules are "adjacent" in a configuration if the cells they occupy are adjacent, or using the notation "$C \setminus \{\mathbf{m}\}$" to denote the configuration obtained by subtracting from C the cell occupied by a module $\mathbf{m}$.

Note that the complement $\overline{C}$ of C might be disconnected, and $\overline{C}$ has exactly one unbounded component. Let ∂C denote the boundary of C and let the *outer boundary* of C be the boundary of the unbounded component of $\overline{C}$. Let $B_{\text{out}}(C)$ be the set of modules in C that have at least one face on the outer boundary. The faces of C in ∂C comprise the *boundary faces* of C. We define the outer boundary of components of $\overline{C}$ in a symmetric fashion; therefore, the boundary faces of the unbounded component of $\overline{C}$ are the same as the boundary faces of C.

A *move* transforms a configuration C into a configuration C' that differs from C by the position of a single module $\mathbf{m}$; we refer to $\mathbf{m}$ as the *moving* module and say that the other modules are *stationary*. Only certain types of moves are considered valid, as we specify next. If for any starting configuration C and target configuration C' there is a sequence of valid moves that can reconfigure C into C', then we say that the model is **universal**; an algorithm that performs such reconfiguration on arbitrary input is likewise called *universal*.

We say that $\mathbf{m}$ is *articulate* in configuration C if the cell occupied by $\mathbf{m}$ is a cut vertex of G_C; $\mathbf{m}$ is *nonarticulate* otherwise. We require the *single backbone condition*: for any configuration C, all valid moves must result in a configuration C' with $C \cap C'$ connected; that is, any valid move must leave the configuration of cells occupied by stationary modules connected. The single backbone condition is equivalent to requiring that only nonarticulate modules move.

In the *sliding model*, two types of moves are allowed: slides and rotations. These *sliding moves* are as follows (refer to Figure 1, where the moving module $\mathbf{m}$ is shown in dark grey):

- A *slide* moves $\mathbf{m}$ from a cell a to an adjacent empty cell b, and requires that there are adjacent occupied cells a' and b' such that a is adjacent to a' and b is adjacent to b'.
- A *rotation* moves $\mathbf{m}$ from a cell a to an empty cell b where a and b share a common edge e, and are both adjacent to an occupied cell c, and requires that the cell $d \notin \{a, b, c\}$ that contains e is empty. Note that every edge e is incident to exactly 4 cells.

We now define a relationship called *slide-adjacency* on the boundary faces of C based on sliding moves. Intuitively, this will allow us to argue about the possible positions that a module $\mathbf{m}$ can occupy after performing a sequence of sliding moves on $C \setminus \mathbf{m}$. We say that a nonarticulate module $\mathbf{m}$ is *attached* at a face f if f is in the common boundary between $\mathbf{m}$ and $C \setminus \{\mathbf{m}\}$. We define a pair of boundary faces f and f' to be *slide-adjacent* if they share an edge and, either (i) both are incident to an empty cell a (Figure 2(a)) or (ii) f is incident to an empty cell a and f' is incident to an empty cell b, and if a module attaches to f, it can move to b with a single sliding move (Figure 2(b)). Two boundary faces f and f'

could share an edge and yet a single sliding move cannot bring a module attached to f to a position where it would be attached to f'. This happens when an edge e is *pinched*, i.e., exactly two of the four cells containing e are occupied and not adjacent (Figure 2(c)).

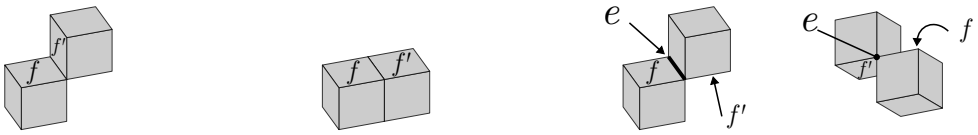


Figure 2 Examples of slide adjacency. (Left) and (center), f and f' are slide-adjacent; (right) shows how two faces f and f' can share an edge but not be slide-adjacent.

The *slide-adjacency graph* of C is the graph representing the slide-adjacency relations on the boundary faces of C. Note that by definition, for every boundary face f and edge e contained in f there is a unique boundary face f' that is slide-adjacent to f and shares e.

3 Reconfiguration Framework

In Section 4 we revisit the $O(n^3)$-move universal reconfiguration algorithm of Abel and Kominers [1] and modify it, showing that it actually performs $O(n^2)$ moves. Thus, we prove the following main theorem:

▶ **Theorem 1.** *Given any two connected unlabeled configurations C and C' each having $n \geq 2$ modules, there exists a reconfiguration of C into C' using $O(n^2)$ sliding moves.*

Similar to the approach of Dimitrescu and Pach [8], we prove our main result by showing that any configuration can be reconfigured into a straight chain of modules, called the *canonical configuration*. This suffices to prove the result, as it follows that any configuration C can be reconfigured into this canonical straight position, and may then be reconfigured into any other configuration C'. Note that the straight configuration may easily be relocated and reoriented in space by rotations and slides. Indeed, in Section 5, we modify the reconfiguration algorithm to be in-place by placing the modules in a more compact form. However, before we can prove Theorem 1 in Section 4, we need several additional definitions and lemmata presented here.

3.1 Structural Properties

In this section, we show some structural properties that are the basis for our algorithms. For ease of description, our figures focus on the case where $d = 3$, but we note that our results are topological and thus extend to higher dimensions. Our analysis ignores the dependency of the dimension in the number of moves (alternatively, it considers that the dimension d is constant).

We also note that the properties listed below assume neither the start nor the end configuration fit in a 2-dimensional plane. Thus, we assume neither configuration fits in a subspace of smaller dimension. If only one configuration fits in a 2-dimensional subspace we virtually perform a single rotation to move a module outside the 2-dimensional space. We use the modified configuration as the initial/target for our algorithm and then undo the move as the first or last step. If both configurations lie in the 2D subspace, we can use a known two-dimensional reconfiguration algorithm (such as [4]).

It is known that any connected graph G on $n \geq 2$ vertices contains at least two distinct non-cut vertices; it follows immediately that any connected configuration C on $n \geq 2$ modules contains at least two non-articulate modules.

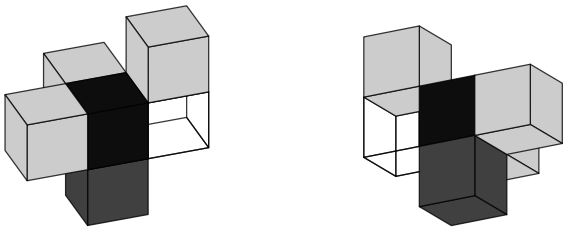


Figure 3 Lemma 2: removing $\mathbf{x}$ (black) disconnects $\mathbf{y}$ (dark gray) from the boundary $B_{\text{out}}(C)$. (Two views are presented.)

▶ **Lemma 2.** *Suppose $\mathbf{x} \in B_{\text{out}}(C)$ is an articulate module, and further suppose that $\mathbf{x}$ is adjacent to a module $\mathbf{y}$ (along face f of $\mathbf{x}$) such that the connected component of $C \setminus \{\mathbf{x}\}$ containing $\mathbf{y}$ is disjoint from $B_{\text{out}}(C)$. Without loss of generality, $\mathbf{y}$ is the bottom neighbor of $\mathbf{x}$. Then, as pictured in Figure 3:*

1. *The face f^{op} of $\mathbf{x}$ opposite f is on the outer boundary of C;*
2. *any module $\mathbf{w} \neq \mathbf{y}$ adjacent to $\mathbf{x}$ is in a component of $C \setminus \{\mathbf{x}\}$ not disjoint from $B_{\text{out}}(C)$; and*
3. *$\mathbf{x}$ is adjacent to at least one such module $\mathbf{w} \neq \mathbf{y}$.*
4. *Moreover, if a cell horizontally adjacent to $\mathbf{x}$ is empty, then the cell directly above it is occupied by a module in $B_{\text{out}}(C)$.*

Proof. Suppose part (1) is false, meaning f^{op} is not on the outer boundary of C. Module $\mathbf{x}$ has some face g on the outer boundary, which is neither f nor f^{op}. Such a g is edge-adjacent to f. Let p be the empty cell adjacent to $\mathbf{x}$ at g, and let q be the cell not containing $\mathbf{x}$ adjacent to both p and $\mathbf{y}$. Since g is on the outer boundary, p is empty. But since $\mathbf{y}$ is not in $B_{\text{out}}(C)$, q must contain a module $\mathbf{m}_q$. However, this means $\mathbf{y}$ is adjacent to $\mathbf{m}_q$, and $\mathbf{m}_q \in B_{\text{out}}(C)$. This contradicts the assumption that the component of $\mathbf{y}$ in $C \setminus \{\mathbf{x}\}$ is disjoint from $B_{\text{out}}(C)$, thus proving part (1).

Now suppose $\mathbf{w} \neq \mathbf{y}$ is adjacent to $\mathbf{x}$ along face h. Let r be the cell adjacent to $\mathbf{x}$ at f^{op}, and let t be the cell adjacent to r and $\mathbf{w}$ not containing $\mathbf{x}$. If t is empty, then clearly $\mathbf{w} \in B_{\text{out}}(C)$. Otherwise, the module $\mathbf{m}_t$ in cell t is adjacent to r (which is empty), so $\mathbf{m}_t \in B_{\text{out}}(C)$. In either case, $\mathbf{w}$ is in a component of $C \setminus \{\mathbf{x}\}$ not disjoint from $B_{\text{out}}(C)$, hence part (2) holds.

Third, since $\mathbf{x}$ is articulate in C, $\mathbf{x}$ has degree at least 2, so it is adjacent to at least one module $\mathbf{w} \neq \mathbf{y}$, proving part (3).

Finally, let e be a cell horizontally adjacent to a face f^e of $\mathbf{x}$. Assume e is empty. Then let e^+ be the cell above e. Now suppose part (4) is false and e^+ is empty as well. Then, f^e must be adjacent to f^{op}, as all faces of $\mathbf{x}$ except the one it shares with $\mathbf{y}$ are adjacent to f^{op}. As e and e^+ are both empty, f^e and f^{op} are slide-adjacent – and hence f^e is also on the outer boundary of C. However, f^e is also slide-adjacent, along its bottom edge, to some face of $\mathbf{y}$ or a face of some neighbor $\mathbf{z}$ of $\mathbf{y}$. In the former case, we have $\mathbf{y} \in B_{\text{out}}(C)$ which contradicts that $\mathbf{y}$ is disjoint from $B_{\text{out}}(C)$. In the latter case, we have $\mathbf{z} \in B_{\text{out}}(C)$ and, since $\mathbf{z}$ is adjacent to $\mathbf{y}$, the component of $C \setminus \{\mathbf{x}\}$ containing $\mathbf{y}$ is not disjoint from $B_{\text{out}}(C)$, another contradiction. Thus, we have part (4). ◀

▶ **Definition 3.** *For a configuration C of n modules, a module $\mathbf{m}$ on $B_{\text{out}}(C)$ is said to be* nearly non-articulate *if $C \setminus \{\mathbf{m}\}$ has exactly two connected components, one of which is disjoint from $B_{\text{out}}(C)$.*

Our algorithm works by moving non-articulate modules on the boundary of C and repeatedly stacking them on some module $\mathbf{s}$. Therefore, we prove that we indeed can find some module $\mathbf{m} \neq \mathbf{s}$ on the boundary that either is non-articulate or can be made non-articulate.

▶ **Lemma 4.** *For any configuration C of $n \geq 2$ modules and a module $\mathbf{s} \in B_{\text{out}}(C)$, there is a module $\mathbf{m} \in B_{\text{out}}(C)$ with $\mathbf{m} \neq \mathbf{s}$ such that $\mathbf{m}$ is either non-articulate or nearly non-articulate in C.*

Proof. As observed at the start of Section 3.1, C contains at least two non-articulate modules; hence, C has at least one non-articulate module $\mathbf{m}_1 \neq \mathbf{s}$. If $\mathbf{m}_1 \in B_{\text{out}}(C)$, then no further argument is required. Otherwise, suppose we have a set $M_{i-1} = \{\mathbf{m}_1, \ldots, \mathbf{m}_{i-1}\} \subseteq C \setminus B_{\text{out}}(C)$ such that for each j with $1 \leq j < i$, the module $\mathbf{m}_j$ is non-articulate in $C \setminus \{\mathbf{m}_1, \ldots, \mathbf{m}_{j-1}\}$. Then $C \setminus M_{i-1}$ is connected, so as before, $C \setminus M_{i-1}$ contains at least one non-articulate module $\mathbf{m}_i \neq s$. Set $M_i = M_{i-1} \cup \{\mathbf{m}_i\}$.

For some minimal $t > 1$, the module $\mathbf{m}_t$ found in this way must be in $B_{\text{out}}(C)$, as there are only finitely many modules in C. If $\mathbf{m}_t$ is a non-articulate module of C, then we have the desired result. Otherwise, by the connectivity of $C \setminus M_t$, all of $B_{\text{out}}(C) \setminus \{\mathbf{m}_t\}$ lies in a single connected component of $C \setminus \{\mathbf{m}_t\}$, so $\mathbf{m}_t$ must have a neighboring cell not in $B_{\text{out}}(C)$. Hence, $\mathbf{m}_t$ must be adjacent to $\mathbf{m}_i$ for some $1 \leq i < t$. By Lemma 2 with $\mathbf{x} = \mathbf{m}_t$, all modules not in the component of $\mathbf{m}_i$ in $C \setminus \{\mathbf{m}_t\}$ are in the component containing $B_{\text{out}}(C) \setminus \{\mathbf{m}_t\}$ (recall that $B_{\text{out}}(C)$ is in a single component), thus removing $\mathbf{m}_t$ leaves exactly two components one of which is disjoint from $B_{\text{out}}(C)$. Hence, $\mathbf{m}_t$ is nearly non-articulate, as required. ◀

▶ **Definition 5.** *A set F of cell faces is* closed *if every edge of a face in F is incident to an even number of faces in F.*

We now show the converse result.

▶ **Lemma 6.** *Every compact set of cell faces F is the boundary of a bounded set of cells.*

Proof. Let c be a cell and $\overrightarrow{r_c}$ be the vertical upwards ray from the center of c. We refer to the *parity* of c as the parity of the number of faces in F that $\overrightarrow{r_c}$ intersects. Let P_F be the set of all cells c with odd parity. Let D be the symmetric difference between ∂P_F and F (viewing ∂P_F as a set of faces). Notice that D is closed since both ∂P_F and F are. We claim that D has no faces parallel to the xy-plane. For the sake of seeking a contradiction, assume face $f \in D$ is horizontal. If $f \in F$, then it is incident to two cells c_1 and c_2 with different parity. Then exactly one of the two is in P_F and $f \in \partial P_F$, a contradiction. Similarly, f cannot be in ∂P_F.

We now claim that D is empty. For contradiction, let f be a highest face in D. Recall that f must not be parallel to the xy-plane. Let e be the top edge of f. By the choice of f, e is not incident to a face in D above it, and D has no horizontal faces. Then, D is not closed, a contradiction. We conclude that $\partial P_F = F$.

It remains to show that P_F is bounded. We show that P_F is contained in the bounding box of F. For contradiction, let $c \in P_F$ be a cell below the bounding box. Let p be the shortest path of adjacent cells connecting c to a cell c' which is not below the bounding box of F. Since c' has even parity, $c' \notin P_F$, and p must have crossed a face f on the boundary of P_F. But $f \notin F$ since it lies below the bounding box. This is a contradiction because $\partial P_F = F$, as proven before. ◀

▶ **Lemma 7.** *Let C be a connected configuration with a connected complement (where the connectedness of the complement is defined as in Section 2). Then, the slide-adjacency graph of C is connected.*

Proof. Let B be a component of the slide-adjacency graph of C. By the definition of slide-adjacency, $V(B)$ is closed, where $V(B)$ represents the vertices of B, i.e., a subset of boundary faces of C. Let P_B be the set of cells described by Lemma 6, with $\partial P_B = V(B)$. We first claim that $P_B \subseteq C$. For contradiction, let c be a cell in $P_B \setminus C$. Then $c \in \overline{C}$ and $\overline{C} \subseteq P_B$ because there are no boundary faces between adjacent cells in $\overline{C}$. Then P_B is unbounded, a contradiction.

We now observe that no cells in P_B are adjacent to cells in C – as otherwise, the face between a pair of such cells would be in the boundary of $P_B \subseteq \partial C$, a contradiction. We then conclude that $P_B = P$, because P is connected. Thus, B is the entire slide-adjacency graph of C. ◀

In the following, we show that once a module reaches the outer boundary, it can reach any other position in the outer boundary without leaving it.

▶ **Corollary 8.** *Let C be a connected configuration. The subgraph of the slide-adjacency graph of C induced by the faces of the outer boundary of C is connected.*

During the execution of our algorithm, we need paths that a module can travel along while avoiding a particular face. The following result allows us to reroute a path in the slide-adjacency graph.

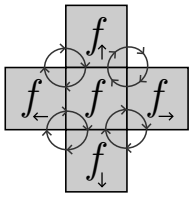


Figure 4 For a face f, with four neighbors $f_\uparrow$, $f_\rightarrow$, $f_\downarrow$, and $f_\leftarrow$, there are four cycles – one per vertex v – which visit f and its two neighbors that are adjacent to v.

▶ **Lemma 9.** *Given a face f with two faces f_1 and f_2 which are slide-adjacent to f there exists a path from f_1 to f_2 in the slide-adjacency graph which does not use f. Moreover, such a path has length $O(1)$ for any fixed dimension d.*

The following is the primary technical tool that allows us to show that we can "free" a module on the boundary of any given configuration.

▶ **Lemma 10.** *Given a configuration C of $n \geq 2$ modules and a module $\mathbf{s} \in B_{\text{out}}(C)$, it is possible to reconfigure C to a configuration C', keeping $B_{\text{out}}(C)$ fixed during the reconfiguration, so that C' has a non-articulate module $\mathbf{x} \neq \mathbf{s}$ in $B_{\text{out}}(C') = B_{\text{out}}(C)$.*

Proof. We induct on n, the number of modules in C. When $n = 2$, both modules are non-articulate and must be in $B_{\text{out}}(C)$, so this is trivially true even without reconfiguration. For the general case, we may find by Lemma 4 a module $\mathbf{x} \in B_{\text{out}}(C) \setminus \{\mathbf{s}\}$ which is either non-articulate or nearly non-articulate in C.

In the former case, $C = C'$, $\mathbf{x}$ is the chosen module and we are done.

In the latter case, $\mathbf{x}$ is nearly non-articulate therefore $C \setminus \{\mathbf{x}\}$ has exactly two components, the "outer" component O and the "inner" component I; specifically, I is the component disjoint from $B_{\text{out}}(C)$, and $O = C \setminus (I \cup \{\mathbf{x}\})$. By Lemma 2(2), there is a unique module

$\mathbf{y} \in I$ adjacent to $\mathbf{x}$. By Lemma 2(3), there is a module $\mathbf{w} \notin I$ adjacent to $\mathbf{x}$, and by Lemma 2(2) $\mathbf{w}$ cannot be opposite from $\mathbf{y}$. So, let c be the cell adjacent to $\mathbf{y}$ and $\mathbf{w}$ not containing $\mathbf{x}$. Cell c must be empty since $\mathbf{w} \notin I$. Let f be the face of $\mathbf{y}$ adjacent to cell c; it is clear that f is on the outer boundary of I (this is a direct consequence of Lemma 2).

Thus, since I has fewer modules than C, the inductive hypothesis shows that we may reconfigure I to I' without moving $B_{\text{out}}(I)$ and then find a non-articulate module $\mathbf{m} \in B_{\text{out}}(I')$ that is distinct from $\mathbf{y}$. It is then enough to show that we can move $\mathbf{m}$ to a position where it is adjacent to both O and I, making $\mathbf{x}$ non-articulate. To do that, we find a path on the outer boundary of $I' \setminus \{\mathbf{m}\}$ to f. If $\mathbf{m}$ can indeed move along this path, then it can reach f and, by its definition, we are done. Else, its movement is obstructed either by $\mathbf{x}$ or by a module in O. We show that, at such position, $\mathbf{m}$ becomes adjacent to both O and I'. In order to prevent a collision between $\mathbf{m}$ and $\mathbf{x}$, we make sure that the path that $\mathbf{m}$ follows does not contain the face f_{xy} shared by $\mathbf{x}$ and $\mathbf{y}$. Note that $I' \setminus \{\mathbf{m}\}$ is connected, and that by Corollary 8, its outer boundary is connected in the slide-adjacency graph. Let f_m be a face to which $\mathbf{m}$ is attached, and note that f_m is on the outer boundary of $I' \setminus \{\mathbf{m}\}$. We can then find a path from f_m to f. If such a path uses f_{xy} we use Lemma 9 to obtain the required path. We note that the fact that such path does not contain f_{xy} is not enough to guarantee that the movement of $\mathbf{m}$ does not intersect $\mathbf{x}$.

It remains to show that when $\mathbf{m}$ cannot move along the computed path $\mathbf{m}$ already connects I' and O. Any two adjacent faces g, g' on the path are connected at either a 90° solid dihedral angle (as in Figure 1(c)), a 180° angle (as in Figure 1(a)), or a 270° angle. Assume that $\mathbf{m}$ is attached to g. If g, g' form a 270° dihedral angle, then $\mathbf{m}$ is already attached to g'. If g, g' form a 180° dihedral angle, $\mathbf{m}$ can perform a slide to attach to g'. If that position is occupied by a module in O then O and I' are already adjacent, a contradiction. If that position is occupied by $\mathbf{x}$ then either $g' = f_{xy}$ (contradicting the definition of the path) or g' is a vertical face which would imply that $\mathbf{x}$ has a neighbor in I' different than $\mathbf{y}$ (contradicting Lemma 2(2)). If g, g' form a 90° dihedral angle, $\mathbf{m}$ can perform a rotation unless the edge shared by g and g' is pinched due to a module not in I'. If it is pinched due to a module in O, then $\mathbf{m}$ is already adjacent to such a module. Similarly, if it is pinched due to $\mathbf{x}$, then by Lemma 2(4) $\mathbf{m}$ is already adjacent to a module in O. ◀

3.2 Reconfiguring into a Canonical Configuration

Using the results from above, we show any configuration C can be reconfigured into a straight chain (a canonical configuration), proving the first part of Theorem 1. That is, any configuration C can be reconfigured into another configuration C'. This result will be the basis of our algorithm that is presented in Section 4.

▶ **Lemma 11.** *Any configuration C can be reconfigured into a straight chain of modules (a canonical configuration).*

Proof. Let $\mathbf{s} \in B_{\text{out}}(C)$ be a module with maximal x_1-coordinate, and let f be the face of $\mathbf{s}$ in the positive x_1-direction. Initially, denote $C_0 = C$ and $Z_0 = \varnothing$. We will iterate, maintaining the following invariants: After step $i-1$ ($1 \leq i \leq n-1$), $\mathbf{s}$ has not moved, and the configuration has the form $C_{i-1} \cup Z_{i-1}$, where Z_{i-1} is a straight chain of $i-1$ modules emanating from face f of $\mathbf{s}$ in the positive x_1 direction, C_{i-1} is connected, and $\mathbf{s} \in B_{\text{out}}(C_{i-1})$.

By Lemma 10, we may reconfigure C_{i-1} to C'_{i-1} while keeping $B_{\text{out}}(C_{i-1})$ fixed in such a way that there is a module $\mathbf{x} \in B_{\text{out}}(C'_{i-1})$ different from $\mathbf{s}$ that is non-articulate in C'_{i-1}. This implies that $\mathbf{x}$ is non-articulate in $C'_{i-1} \cup Z_{i-1}$. By Corollary 8, the subgraph of the

slide adjacency graph induced by the outer boundary of $C'_{i-1} \cup z_{i-1}$ is connected so we may move $\mathbf{x}$ along the boundary of $C'_{i-1} \cup Z_{i-1} \setminus \{\mathbf{x}\}$ so that it extends the chain Z_{i-1}. Let Z_i be this new chain of length i, and let C_i be $C'_{i-1} \setminus \{\mathbf{x}\}$. These clearly satisfy the required invariants.

After stage $n-1$, the reconfiguration is complete. ◀

4 LocateAndFree-based Algorithm

The proof given in Section 3.2 gives rise to a simple algorithm to reconfigure an n-module configuration C into a straight chain. Here we present this algorithm (Algorithm 2) and prove its correctness. We first require a recursive method that, given a configuration C and a module $\mathbf{s} \in B_{\text{out}}(C)$ (along with a particular face of s on the outer boundary), modifies C and returns a module $\mathbf{x}$ according to Lemma 10. We assume that each module $\mathbf{m}$ has previously been assigned a attribute **PostOrder**($\mathbf{m}$) which sorts the modules of C in the order of finishing times of a depth-first search in G_C beginning at $\mathbf{s}$.

See Algorithm 1, which converts Lemma 10 to a routine LocateAndFree. Most of Algorithm 1 follows Lemma 10 directly. To prove Algorithm 1 correct, we must address the comments in lines 3 and 9.

Algorithm 1 Locate a module $\mathbf{x} \in B_{\text{out}}(C)$ satisfying Lemma 10, and reconfigure the interior of C to render $\mathbf{x}$ non-articulate. Assumes **PostOrder** attributes in C have been set.

```
1:  LocateAndFree(C, s) :=
2:    Locate all faces in the outer boundary of C by DFS from s. We obtain B_out(C).
3:    Choose x ∈ B_out(C) with smallest PostOrder. {x is (nearly) non-articulate}
4:    Compute all modules in the component O of C \ {x} containing s by DFS.
5:    if O contains all neighbors of x then
6:      return x.
7:    else
8:      Let y be the neighbor of x in the other component I := C \ (O ∪ {x}).
9:      Let m = LocateAndFree(I, y). {Use existing PostOrder labels.}
10:     Move m to connect O and I as in Lemma 10, locating its path by DFS across the
        outer boundary of C \ {m}.
11:     return x.
12:   end if
13: end LocateAndFree
```

First, we must show that the module $\mathbf{x} \in B_{\text{out}}(C)$ with minimal **PostOrder** is non-articulate or nearly non-articulate. If $\mathbf{x}$ is articulate in C, then a path from $\mathbf{s}$ to any module $\mathbf{t} \notin O$ must pass through $\mathbf{x}$, meaning

$$\textbf{PostOrder}(t) \leq \textbf{PostOrder}(x).$$

This means $\mathbf{t}$ cannot be in $B_{\text{out}}(C)$, by the minimality of **PostOrder**($\mathbf{x}$). Thus, any connected component of $C \setminus \{\mathbf{x}\}$ not containing $\mathbf{s}$ is disjoint from $B_{\text{out}}(C)$, so Lemma 2 applies, proving that $\mathbf{x}$ is indeed nearly non-articulate.

We must also prove that the attribute **PostOrder** sorts the modules of I in a post-order from $\mathbf{y}$. By choice of $\mathbf{x}$, the original depth-first tree restricted to I must itself be a valid depth-first tree of I rooted at $\mathbf{y}$, and thus the **PostOrder** field is correctly sorted, as needed.

Now we present Algorithm 2, which rearranges C into a straight chain, using Algorithm 1 as a subroutine. The proof of correctness of Algorithm 2 follows directly from the results in Section 3.2.

Algorithm 2 Reconfigure C into a straight chain $\{\mathbf{s}\} \cup Z_{n-1}$.

1: Fix a module $\mathbf{s} \in C$ with maximal x_1-coordinate.
2: Set $C_0 = C$ and $Z_0 = \varnothing$.
3: **for** $1 \leq i < n$ **do**
4: Set the **PostOrder** fields with a depth-first search rooted at $\mathbf{s}$.
5: Define $\mathbf{x} :=$ LocateAndFree$(C_{i-1}, \mathbf{s})$.
6: By depth-first search across the outer boundary faces of $C_{i-1} \cup Z_{i-1} \setminus \{\mathbf{x}\}$, move $\mathbf{x}$ to extend Z_{i-1}. Define $C_i := C_{i-1} \setminus \{\mathbf{x}\}$ and $Z_i = Z_{i-1} \cup \{\mathbf{x}\}$.
7: **end for**

4.1 Algorithm Analysis

We first briefly comment on the analysis from [1]: Each of the $n-1$ iterations of Algorithm 2 may in principle make $O(n)$ recursive calls to LocateAndFree. In each recursive call, $\mathbf{m}$ moves $O(n)$ times; thus, the overall number of moves is $O(n^3)$. Relative to [1], we are more careful in how we define the path in the proof of Lemma 10. In [1], Abel and Kominers use a similar induction on C, taking the interior component I of C, and reconfiguring it into I'; they then define their path on the outer boundary of I'. The Abel and Kominers [1] analysis gave no proof of connectivity of the outer boundary of I'. Additionally $\mathbf{x} \notin I'$, so the face f_{yx} that $\mathbf{y}$ shared with $\mathbf{x}$ is on the boundary of I' even though it is actually inaccessible as $\mathbf{x}$ is attached to f_{yx}, so this path would need to avoid collision with $\mathbf{x}$. Finally, since the path is defined on the outer boundary of I and not $I' \setminus \{\mathbf{m}\}$, it would be possible that the path found uses a face adjacent to the initial position of $\mathbf{m}$, which no longer exists as $\mathbf{m}$ is the mobile module, hence the path needs to avoid $\mathbf{m}$'s initial position as well. The latter two issues were also unaddressed in [1].

We now present an improved analysis of the algorithm, addressing the connectivity issue along the way. We show that although LocateAndFree may recursively call itself $O(n)$ times, it only uses $O(n)$ moves over the entire execution of the algorithm. Therefore we can use LocateAndFree to make a single module on the boundary non-articulate with $O(n)$ moves. For this, we need to assume that the path computed in Lemma 10 is the *shortest* path from f_m to f while avoiding f_{xy} – this can be computed with breadth-first search after deleting f_{xy} from the slide adjacency graph of $I' \setminus \{\mathbf{m}\}$.

▶ **Lemma 12.** *LocateAndFree$(C, \mathbf{s})$, with all its recursive calls, executes $O(|I|)$ moves in any fixed dimension, where $|I|$ is the number of modules in the configuration I as defined in Algorithm 1.*

Proof. We use induction on the number of recursive calls. If LocateAndFree$(\mathrm{C}, \mathbf{s})$ does not make any recursive call, then no move is performed. We consider the recursive call LocateAndFree$(I, \mathbf{y})$ in line 9 of Algorithm 1. Let $O_1, I_1, \mathbf{x}_1 = \mathbf{m}$ and $\mathbf{m}_1$ play the role of $O, I, \mathbf{x}$ and $\mathbf{m}$ for the recursive call, respectively. Thus $O_1 \cup I_1 \cup \{\mathbf{m}\} = I$. By inductive hypothesis, LocateAndFree$(I, \mathbf{y})$ performs $O(|I_1|)$ moves, transforming I into a configuration I' with the same number of modules. It indeed only changes I_1 into I_1', maintaining O_1 by construction. It is then enough to show that $\mathbf{m}$ moves at most $O(|O_1|)$ times. The general structure of $O, I, O_1, I_1, \ldots$ are depicted in Figure 5.

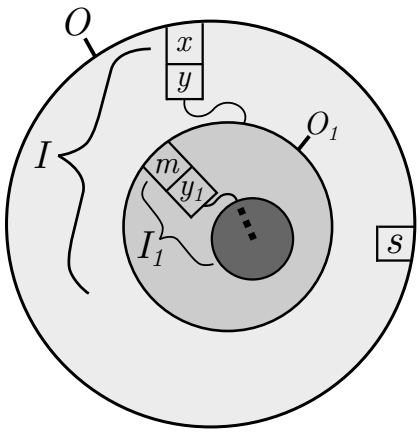


Figure 5 O_i and I_i are connected by one module; O_{i+1} and I_{i+1} are a subset of I_i.

Recall that $\mathbf{m}$ moves on the outer boundary of $I' \setminus \{\mathbf{m}\}$. Thus it moves on the outer boundary of $O_1 \cup I'_1$. By choice of $\mathbf{x}_1 = \mathbf{m}$, f lies on the boundary of O_1. By Lemma 2(3), if $|I_1| \neq 0$, m is adjacent to a module in $B_{\text{out}}(O_1)$ of which one face is f_m. Without loss of generality, assume the top face of $\mathbf{m}$ is on the outer boundary of I'. Note that $B_{\text{out}}(I') = B_{\text{out}}(O_1 \cup \{\mathbf{m}\} \cup I'_1) = B_{\text{out}}(O_1 \cup \{\mathbf{m}\})$, i.e., I'_1 is completely "inside", only getting exposed when we delete $\mathbf{m}$. Thus, the four faces that are slide-adjacent to the top face of $\mathbf{m}$ are the only faces of ∂O_1 that are slide-adjacent to faces of $\partial I'_1$ in the outer boundary of $O_1 \cup I'_1$. Note that f_m is slide adjacent through its top edge to at least one of these four faces. The path taken by $\mathbf{m}$ might use faces of I'_1 but it eventually enters O_1 for the last time to reach f, thus passing through one of these four faces. By Lemma 9, the distance between these four faces is $O(d)$. Thus, the path uses $O(d)$ faces of I'_1, and the path length is $O(|O_1|)$ given constant dimension. ◀

5 Bounded-Space Algorithm

LocateAndFree can be used to design a different reconfiguration algorithm with two significant properties. First, the algorithm is *in-place* – during the whole reconfiguration process, stationary modules will be contained in the bounding box of the union of the start and end configuration (plus possibly a small $O(1)$ margin). Second, the algorithm is *input-sensitive* – the number of moves needed is bounded by the number of modules n and the volume of the start and end configurations.

Due to size constraints, a description of the algorithm has been omitted but can be found in the full version of this paper (`https://arxiv.org/abs/0802.3414`). The general idea is to use LocateAndFree to free a number of modules, which build a "scaffolding" around the configuration, making it easy to compact into a parallelogram, which is the new canonical configuration.

Let B be the bounding box of the union of the start and end configurations with dimensions $x_M \times y_M \times z_M$. By translation and coordinate reflection, we can assume that one of the corners of B is the origin and that its opposite corner is (x_M, y_M, z_M). Moreover, by renaming the axis, we can also assume that $2 \leq x_M \leq y_M \leq z_M$. With these definitions, we can state the Bounded-Space algorithm:

▶ **Theorem 13.** *Given any two connected unlabeled configurations C and C' each having $n \geq 2$ modules, there exists an in-place reconfiguration of C into C' using $O(n \cdot \min\{n, x_M y_M + z_M\})$ sliding moves.*

6 Conclusions and Future Work

The algorithms presented in this paper raise our understanding of the 3D (and higher) sliding cube configuration problem to a level comparable to the 2D counterpart. Although our algorithms are optimal from several points of view (maximum required number of moves, workspace size and input-sensitiveness), several issues remain open:

- Due to the practical constraints of physical robots it is desirable for these algorithms to run in a distributed fashion using mostly local information. Although LocateAndFree transforms connectivity questions into relatively cheap postorder count comparisons, currently this information cannot be easily updated after a module has been freed. Can the update be somehow localized?
- The key property for LocateAndFree is Lemma 2. Since the result is topological, it applies to any dimension $d > 2$. Although it would not make much sense to extend the results to higher dimensions, it would be interesting to explore if there are other meaningful module shapes for which the same lemma (possibly with minor variations) applies.
- The current Bounded-Space algorithm compares n to $x_M y_M$ and either uses one slab or none (see Appendix). This leads to a serial algorithm, it would be interesting to modify the algorithm to be parallelizable – say, construct c equally spaced slabs (for some $c = c(n, x_M y_M, z_M)$), and have each one be responsible for a fraction of the domain. This would not impact the total number of moves but would reduce the *makespan* required to execute them.
- As stated in Section 1, sliding cube algorithms can be applied to the crystalline model via $2 \times 2 \times 2$ *meta-modules* (sub-arrangements of modules that are treated as atomic units). Are there meta-modules for other models of modular robots that emulate sliding cubes and can therefore use our algorithms for universal configuration?
- Akitaya et al. [4] proved it was NP-hard to compute the shortest reconfiguration sequence in the 2D sliding square model. Can their result be extended to higher dimensions?

References

1 Zachary Abel and Scott D. Kominers. Universal reconfiguration of (hyper-)cubic robots. *arXiv preprint*, 2008. arXiv:0802.3414v3.

2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ Musketeers. *Algorithmica*, 83:1316–1351, 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 Kevin Buchin and Éric Colin de Verdière, editors, *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 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 Artur Czumaj and Qin Xin, editors, *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2022.4.

5 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D Demaine, Dania El-Khechen, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Val Pinciu, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Realistic reconfiguration of crystalline (and telecube) robots. In *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, pages 433–447. Springer, 2009. doi:10.1007/978-3-642-00312-7_27.

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 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using $o(\log n)$ parallel moves. In *Proceedings of Algorithms and Computation: 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008*, pages 342–353. Springer, 2008. doi:10.1007/978-3-540-92182-0_32.

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

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

10 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, pages 2460–2467. IEEE, 2003. doi:10.1109/IROS.2003.1249239.

11 Irina Kostitsyna, Tim Ophelders, Irene Parada, Tom Peters, Willem Sonke, and Bettina Speckmann. Optimal in-place compaction of sliding cubes. *arXiv preprint*, 2024. arXiv:2312.15096.

12 Tillmann Miltzow, Irene Parada, Willem Sonke, Bettina Speckmann, and Jules Wulms. Hiding sliding cubes: Why reconfiguring modular robots is not easy (media exposition). In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.78.

13 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proceedings of the 36th European Workshop on Computational Geometry (EuroCG'20)*, 2020. Art. 32.

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