Sliding Squares in Parallel

Hugo A. Akitaya
Miner School of Computer and Information Sciences, University of Massachusetts Lowell, MA, USA

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany

Peter Kramer
Department of Computer Science, TU Braunschweig, Germany

Saba Molaei
Sharif University of Technology, Teheran, Iran

Christian Rieck
Department of Discrete Mathematics, University of Kassel, Germany

Frederick Stock
Miner School of Computer and Information Sciences, University of Massachusetts Lowell, MA, USA

Tobias Wallner
Department of Computer Science, TU Braunschweig, Germany

Abstract

We consider algorithmic problems motivated by modular robotic reconfiguration in the *sliding square model*, in which we are given n square-shaped modules in a (labeled or unlabeled) start configuration and need to find a schedule of sliding moves to transform it into a desired goal configuration, maintaining connectivity of the configuration at all times. Recent work has aimed at minimizing the total number of moves, resulting in fully sequential schedules that can perform reconfiguration in $\mathcal{O}(n^2)$ moves, or $\mathcal{O}(nP)$ for arrangements of bounding box perimeter size P.

We provide first results in the sliding square model that exploit parallel motion, performing reconfiguration in worst-case optimal makespan of $\mathcal{O}(P)$. We also provide tight bounds on the complexity of the problem by showing that even deciding the possibility of reconfiguration within makespan 1 is NP-complete in the unlabeled case. In the labeled variant, we note that deciding the same for makespan 2 is NP-complete, while makespan 1 is straightforward.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases Sliding squares, parallel motion, reconfigurability, motion planning, multi-agent path finding, makespan, swarm robotics, computational geometry

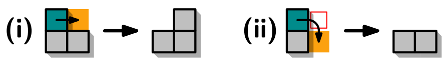
Digital Object Identifier 10.4230/LIPIcs.ESA.2025.28

Related Version *Full Version*: https://arxiv.org/abs/2412.05523 [5]

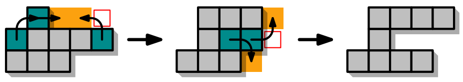
Funding This work was partially supported by the German Research Foundation (DFG), project "Space Ants" (FE 407/22-1) and grant 522790373, by the National Science Foundation (NSF), grant CCF-2348067, and by a fellowship of the German Academic Exchange Service (DAAD).

1 Introduction

Reconfiguring an arrangement of geometric objects is a fundamental task in a wide range of areas, both in theory and practice. 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 [9, 28, 30] has enjoyed particular attention [1, 2, 3, 4, 6] in the context of Computational Geometry: In the *sliding square model* introduced by Fitch, Butler, and Rus [19], a given start configuration of n identical 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.



(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 occur in parallel, collision-free transformations. In this paper, we show the symmetric difference of a transformation using turquoise and yellow.

Aiming at minimizing the total number of moves, previous research has resulted in considerable progress, recently establishing [2] universal configuration in $\mathcal{O}(nP)$ moves for a 2-dimensional arrangement of n modules with bounding box perimeter size P, and $\mathcal{O}(n^2)$ in three dimensions [1, 22]. However, the resulting schedules are purely sequential, not optimally minimizing the overall time until completion, called *makespan.* With parallel motion, much lower makespan can be achieved – which is also a more challenging objective, as it requires coordinating the overall motion plan, not just at the atomic level (see Figure 1(b)), but also at the global level to maintain connectivity and avoid collisions.

1.1 Our contributions

We provide first results for *parallel* reconfiguration in the sliding squares model with no restriction on the input. We achieve tight outcomes, both on the negative and the positive side.

1. We prove that even deciding the existence of a schedule with the smallest possible makespan of 1 is NP-complete in the parallel reconfiguration model, as well as APX-hard.
2. We give an 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. We show that this is asymptotically worst-case optimal.

Due to limited space, we only provide high-level descriptions here and refer to the full version [5] for the majority of our proofs and full technical details. Additionally, we extend our results to the *labeled* variant: When individual robots are distinguishable, deciding makespan 2 is NP-complete, while makespan 1 is easy. Moreover, asymptotically worst-case optimal schedules can still be computed in polynomial time with a relaxation of the in-place requirement. For precise definitions and the problem statement, see Section 2.

1.2 Related work

On the theoretical side, algorithmic methods for coordination the motion of many robots can be traced back to the classical work of Schwartz and Sharir [29] from the 1980s. On the practical side, [20] presented an architecture for modular robots, followed by a wide spectrum of work that often used cuboids as elementary building blocks; see [32] for a survey.

Of particular interest for the algorithmic side is the *sliding cube model* (or *square*) by Fitch, Butler, and Rus [19], introduced in the context of a modular robotic hardware [9, 28, 30]. Recent work [1, 4, 13, 22] has studied algorithmic methods for *sequentially* sliding squares and cubes, with typical schedules requiring a quadratic number of moves. Also related are models with slightly different types of moves, such as "pivoting", see [2, 3, 10, 24].

In [24] the authors claim an algorithm for connectivity-preserving reconfiguration of sliding squares in $\mathcal{O}(n)$ parallel steps. Unfortunately, this is incorrect for general input. Nevertheless, if the input configurations are guaranteed to not contain a specific "forbidden" local pattern (a 2×2 arrangement with only two diagonally adjacent squares), then their algorithm is correct. Similar models of parallel sliding squares have been investigated experimentally [31]. However, this still leaves the existence of linear-time parallel protocols unresolved.

While the previous work on sliding squares focuses on minimizing the number of moves, a different line of research aims at minimizing the total time until completion. In [21], the authors study parallel distributed algorithms for a less restrictive model of sliding squares, obtaining $\mathcal{O}(n)$ makespan. They show how to obtain the same result in the standard sliding square model using *meta-modules*, small groups of cooperating modules that behave like a single entity. This, however, imposes extra constraints in the input. Algorithmic research on meta-modules was considered in [6, 27]. In [7, 12], the authors considered coordinated motion planning in a less constrained model (see discussion in Section 2) for labeled robots to minimize the makespan, aiming for *constant stretch*, i.e., a makespan within a constant factor of the maximum distance for a single robot. This was extended to preserve connectivity for unlabeled [8, 16] and labeled robots [18], and between obstacles [17].

Scaffolding and Meta-modules. Computing reconfiguration paths is not a simple task and, for many models, it is computationally intractable (even PSPACE-complete [3]). *Scaffolding* (proposed independently by [23] and [26]) aims at making reconfiguration simpler by restricting the scope to configurations containing a regular substructure (the *scaffold*) that remains connected between moves (thus removing complexity from the connectivity constraint) while having enough empty positions to allow modules to move through the scaffold (thus removing complexity from the free space constraints). They obtain a scaffold by using *meta-modules*, groups of modules that cooperate to move as a higher-level functional unit. Previous work has used these to translate algorithms for a particular model to another (see, e.g., [21, 27, 28]).

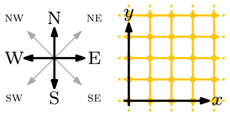
The drawback of these techniques is that their input must belong to the subset of configurations where modules are already organized into meta-modules (the scaffold exists in both the initial and target configuration). The authors of [8, 16] proposed an algorithm that first builds the scaffold from an arbitrary scaled configuration. They achieve *constant stretch*, i.e., a makespan within a constant factor of the maximum minimum distance for a single robot (based on previous work [7, 12]). This was extended for labeled modules [18], and between obstacles [17]. Although their techniques apply to a wider range of configurations, there are still restrictions on the input. The algorithm presented in this paper also makes use of scaffolding and meta-module techniques. To the best of our knowledge, our algorithm is the first universal one to build the scaffold and meta-modules (i.e., from arbitrary input).

Model comparison. Our model is more constrained than existing models for parallel reconfiguration under connectivity constraints [15, 16, 18, 21, 24]. Michail, Skretas, and Spirakis [24] do not explicitly require the connected backbone constraint, and do not explicitly define the collision model. The models in [16, 18] do not require the connected backbone constraint, and allow modules to enter and leave cells simultaneously, even in orthogonal directions. Similarly, [21] relaxes the free-space constraint of the convex transition move, allowing a module to move through two diagonally adjacent static modules. However, the authors disallow chain moves as shown in Figure 1(b), so that the cells involved in individual slide moves in the same transformation are disjoint. The same is true for [14, 15] where the paths of transformations at any given time stamp must be disjoint.

2 Preliminaries

We study the connected reconfiguration of squares in the infinite, two-dimensional integer grid by sliding moves in a parallel variant of the sequential *sliding squares* model, as follows.

Configurations. Each simple 4-cycle of edges in the infinite integer grid $\mathbb{Z}^2$ bounds a unit *cell* c, which is uniquely identified by its minimal integer coordinates $x(c)$ and $y(c)$. Two such cells are *edge-adjacent* (resp., *vertex-adjacent*) if their boundary cycles share an edge (resp., vertex). A *configuration* C of n square *modules* is defined by the set of occupied cells, each containing a single module. We say that C is valid if and only if its dual graph, defined by edge-adjacency, is connected, and denote by B the unique axis-aligned bounding box of minimal perimeter P that contains C; see Figure 2 for illustrations.



(a) The integer grid $\mathbb{Z}^2$.

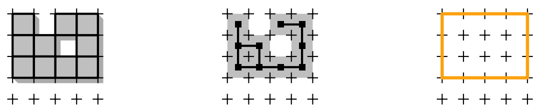

(b) A configuration C of 10 modules, its dual graph, and bounding box B.

Figure 2 We illustrate the relation of the integer grid and configurations.

Moves. Individual modules can perform two types of *move* into an unoccupied edge-adjacent or vertex-adjacent cell, *slides* and *convex transitions*, as illustrated in Figure 1(a). We denote a move by its start and target cells, e.g., $u \rightsquigarrow v$. In our model, both moves take an identical (unit) duration to complete, which allows us to easily extend the existing notion of move counting in sequential models [4, 13, 15, 25] to the parallel setting. Note that in our figures, the path denoting a convex transition is shown containing a circular arc for clarity. Such a path would be accurate if the corners of modules were rounded.

Transformations. The parallel execution of moves constitutes a *transformation*, which we denote by the initial and resulting configurations, e.g., $C_1 \to C_2$. A transformation is valid only if it (i) preserves connectivity and (ii) does not cause collisions.

As in a number of existing models for both sequential and parallel reconfiguration of squares in the grid [1, 4, 22, 31], connectivity is determined by the existence of a *connected backbone*: Given a configuration C, we call a subset M of modules *free* if $C \setminus M'$ is a valid configuration for any $M' \subseteq M$. Let $M \subset C_1$ refer to all moving modules during $C_1 \to C_2$. Then there exists a connectivity-preserving backbone if and only if M is free both in C_1 and in C_2. A module m is *free* if $\{m\}$ is free. In Figure 3, both transformations are illegal as they attempt to move modules that are not free (conflicts are marked in red).

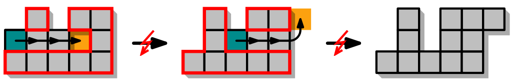


Figure 3 Moving non-free modules causes disconnections and makes transformations illegal.

Secondly, a transformation is collision-free exactly if all modules can move along their designated path at a constant rate, completing it in unit time without overlapping with any other module in the process. We illustrate a partial selection of collisions in Figure 4. Similar to the sequential variant, the defined moves and transformations are reversible, i.e., if for any transformation $C_1 \to C_2$, there exists an inverse transformation $C_2 \to C_1$.

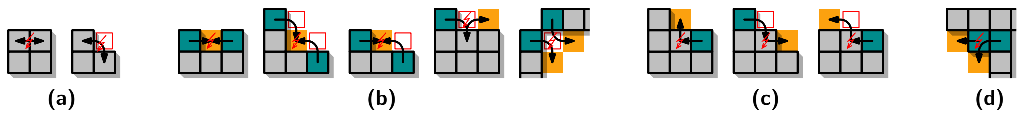


Figure 4 Collisions: (a) Modules cannot swap places or (b) enter the same cell simultaneously, (c) moves cannot meet at endpoints orthogonally, and (d) speed mismatches must be avoided.

Problem statement. The Parallel Sliding Squares problem, asks for the connected reconfiguration of modules in the infinite integer grid by legal transformations, with an *instance* $\mathcal{I}$ being composed of two configurations (C_1, C_2) of n modules. A schedule is *feasible* for $\mathcal{I}$ if and only if it transforms C_1 into C_2. The *makespan* of a schedule is the number of transformations in it. The decision problem for Parallel Sliding Squares asks, given an instance $\mathcal{I}$ and $k \in \mathbb{N}^+$, whether there is a feasible schedule for $\mathcal{I}$ with makespan at most k.

Labeled problem variant. In the case of distinguishable (or *labeled*) modules, we speak of the Labeled Parallel Sliding Squares problem. This variant uses the exact same move set and collision constraints as the unlabeled variant, but configurations are now mappings between $[n]$ and $\mathbb{Z}^2$ rather than subsets thereof. This allows us to specify individual target positions and track the movement of individual modules across transformations.

Useful notation. Throughout this paper, we make use of cardinal and ordinal directions. In particular, the unit vector $(1, 0)$ points *east*, $(0, 1)$ points *north*, and their opposite vectors point *west* and *south*, respectively. For an illustration, see Figure 2(a). Throughout this paper, we use B_1 and B_2 to denote the bounding boxes of C_1 and C_2, respectively. We assume, as in the literature [25], that B_1 and B_2 share a south-west corner at $(0, 0)$.

We define the (open) neighborhood $N(c)$ of a cell c as the set of cells that are edge-adjacent to c, and the closed neighborhood $N[c]$ as $N(c) \cup \{c\}$. We abuse notation to express neighborhoods of sets of cells as $N[S] = \bigcup_{c \in S} N[c]$ and $N(S) = N[S] \setminus S$. Analogously, define $N^*(S)$ (resp., $N^*[S]$) as the open (resp., closed) neighborhood using vertex-adjacency.

A feasible schedule for $\mathcal{I}$ is (*strictly*) *in-place* (as defined in [4, 25]) if and only if no intermediate configuration exceeds the union $B_1 \cup B_2$ by more than one module. We relax this constraint slightly and say that a schedule is *weakly in-place* if no intermediate configuration exceeds the union $B_1 \cup B_2$ by more than a constant number of units. Note that we do not restrict the number of modules located outside the union, only their distance to it. This corresponds to a slightly more restricted variant of the definition used in [1].

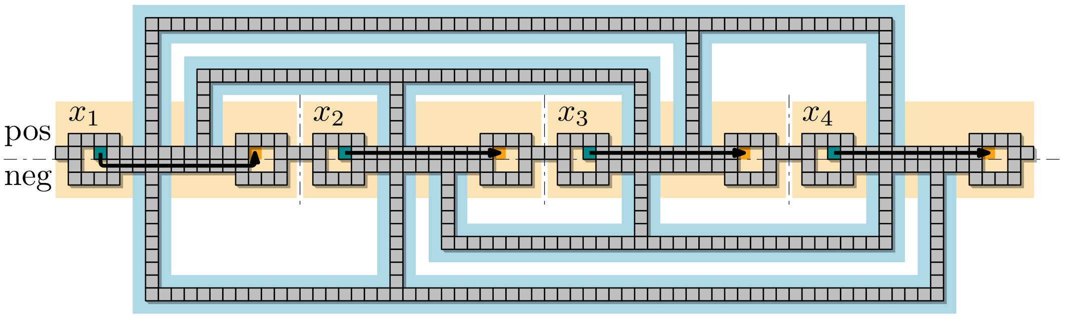
3 Computational complexity

We provide a number of complexity results for Parallel Sliding Squares, which are complementary to the NP-completeness result obtained by the authors of [4] for the sequential variant of this problem. This highlights a previously unrecognized gap in complexity, compared to closely related models for parallel, connected reconfiguration without the connected backbone constraint. In particular, in the related model studied in [16, 18], deciding the existence of schedules of makespan 2 is NP-complete, while makespan 1 is in P. We give a high-level description of our construction for the unlabeled case and refer to the full version [5] for technical details, as well as the labeled problem variant.

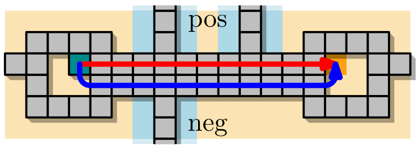
▶ **Theorem 1.** *Let $\mathcal{I}$ be an instance of Parallel Sliding Squares. It is* NP*-complete to decide whether there exists a feasible schedule of makespan* 1 *for $\mathcal{I}$.*

We reduce from the NP-complete problem Planar Monotone 3Sat [11], which asks whether a Boolean formula in conjunctive normal form is satisfiable. Each clause consists of at most 3 literals, all either positive or negative, and the clause-variable incidence graph must admit a plane drawing where variables are mapped to the x-axis, positive (resp., negative) clauses are mapped to the upper (resp., lower) half-plane, and edges do not cross the x-axis.

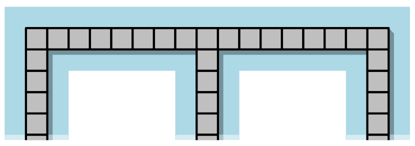
Our reduction starts with a rectilinear embedding of the clause-variable incidence graph of an instance φ of Planar Monotone 3Sat and constructs an instance $\mathcal{I}_\varphi$ of Parallel Sliding Squares using *variable* and *clause gadgets*. We then argue that φ can be satisfied if and only if there is a single parallel transformation that solves $\mathcal{I}_\varphi$. For a formula φ over m variables with k clauses, we create m copies of the variable gadget in a horizontal line, directly adjacent to one another. The literals in each monotone positive (negative) clause are then connected to by a clause gadget placed above (below) the horizontal line of variables. A simple example of our construction is depicted in Figure 5(a).



(a) 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 $\alpha(\varphi) =$ (`true`, `false`, `false`, `false`).



(b) Variable gadget (blue/red ≡ `true`/`false`).



(c) Clause gadget.

Figure 5 An overview of our hardness reduction and the two types of gadget used.

Variable gadget. The variable gadget consists of two cycles of 12 modules each, the left containing an additional module in its interior, see Figure 5(b). These circles are connected by a solid, horizontal *assignment strip* of height 2, to which individual clause gadgets (blue) are connected. In the target configuration, the extra module is located in the other circle. There are two feasible transformations, each representing either a `true` or `false` value assignment.

Clause gadget. The clause gadget consists of a thin horizontal strip that spans the gadgets of variables contained in the clause, and (up to) three vertical prongs that connect to the assignment strips of the incident variable gadgets. No squares move here, see Figure 5(c).

Due to the way in which both configurations are connected, the schedules depicted in Figure 5(b) locally sever the direct connection between the variable gadget and either all its positive or negative clauses. Thus, a satisfying assignment maps to a feasible transformation.

▶ **Corollary 2.** *Parallel Sliding Squares is* APX*-hard and cannot be approximated within a factor better than* 2 *in polynomial time unless* P = NP.

4 A worst-case optimal algorithm

In this section, we introduce a three-phase algorithm for efficiently reconfiguring two given configurations into one another. 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.

▶ **Theorem 3.** *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 three phases as depicted in Figure 6: In **Phase (I)**, we identify a subconfiguration used as a "backbone" (similar to [21]), 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 modules into grid-aligned 3×3 squares, creating a 3-scaled configuration. In general, a configuration is *c-scaled* exactly if it is composed of $c \times c$ squares that are aligned with a corresponding $c \times c$ integer grid. By generalizing techniques from [16], we can reconfigure these efficiently:

▶ **Theorem 4.** *For any 3-scaled instance $\mathcal{I}$ of* Parallel Sliding Squares*, we can compute a feasible, in-place schedule of $\mathcal{O}(P_1 + P_2)$ transformations in polynomial time.*

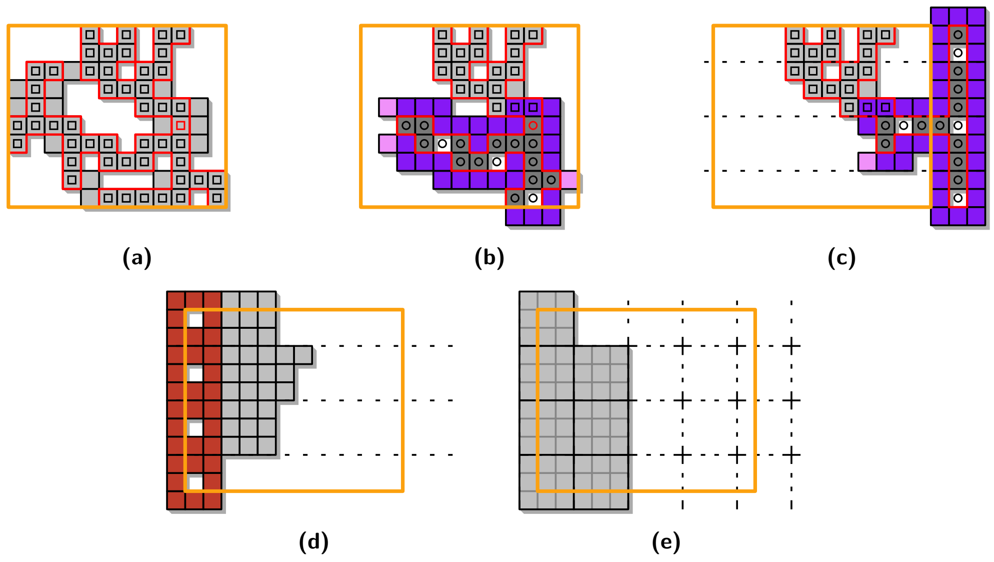


Figure 6 The high-level overview of our approach. (a) Initial configuration and (b), (c) and (e) are the result of Phases **(I–III)**, respectively. (d) shows an intermediate configuration in Phase **(III)**.

To reach our target configuration, we then simply apply **Phases (I-III)** in reverse. Although several of our techniques are inspired by previous work, they differ substantially in our context of parallel reconfiguration and considerable changes were needed.

▶ **Lemma 5.** *The bounds achieved in Theorem 3 are asymptotically worst-case optimal.*

Proof. Assume that n is even and consider two configurations C_1, C_2 contained in an $n/2 \times n/2$ bounding box B. Define C_1 to contain the $n-1$ modules adjacent to the left and bottom edges of B and a module at $(1, 1)$. Define C_2 to contain the $n-1$ modules adjacent to the top and right edges of B and a module at $(n/2 - 1, n/2 - 1)$. Then, the minimum bottleneck matching between full cells in C_1 and C_2 has bottleneck $\Omega(n)$, implying that a module must make $\Omega(n)$ moves. The makespan is then $\Omega(n) = \Omega(P)$, where P is the perimeter of B. ◀

4.1 Phase (I): Gathering squares

In **Phases (I)** and **(II)**, we use an underlying connected substructure (called *skeleton*) of the initial configuration to guide the reconfiguration. Intuitively (formal definitions below), this tree-like skeleton functions as a backbone around which we move modules toward a "root" module, making a subtree "thick" (where the cells in the neighborhood of part of the skeleton are all full). This "thick" subskeleton is more "manipulable", making it easier to mold it into a sweep line in **Phase (II)**. Moving modules around a tree is an idea also used by the authors of [21]. This type of strategy becomes complicated when the tree creates bottlenecks where potential collisions happen, which they solve by strengthening the model and allowing modules to "squeeze through" bottlenecks. We achieve a stronger result in the classic sliding model via careful definition of the tree-like structure and movement schedules.

Skeleton. We define the *skeleton* of a configuration C as a valid subconfiguration S such that: $C \subset N[S]$ (every module of C is either contained in S or edge-adjacent to a module of S); and cycles in the dual graph of S are pairwise disjoint with length at most 4. We call a module in S (resp., not in S) a skeleton (resp., nonskeleton) module. In Figure 6(a), skeleton modules are highlighted with an internal square and the perimeter of the skeleton is shown in red. Note that any set of nonskeleton modules is free.

On a high level, we can compute a skeleton S of a given configuration C as follows. Think of S as a subset of C, initialized as the empty set. Modules of C are then algorithmically added to S until it satisfies the definition of a skeleton. First, all modules with even x-coordinate are added to S, then all modules with odd x-coordinate with no east and west neighbors. Next modules are added to S until it is a connected subconfiguration of C (see magenta squares in Figure 6(a)). This might introduce large cycles (greater than length 4) to S. These cycles are broken via removal of modules from S or exchanging adjacent modules. We show that careful execution of these steps will build S into a skeleton of C.

Equipped with a skeleton S, we now consider its dual graph. Contracting every cycle in the dual of S to a single vertex renders a max-degree-4 tree where each node is either a cell or a 4-cycle. We refer to the nodes of this tree as *nodes of S* and abuse notation by referring to S as a tree. Let r be the *root* node of S, arbitrarily chosen. For every nonskeleton module, we assign an adjacent module of S as its *support.* We define the *subskeleton rooted at c* (denoted S_c) as the subconfiguration containing c and its descendant nodes. The set S_c^* contains the modules in S_c and the nonskeleton modules supported by modules in S_c. The *weight* of a subskeleton S_c is defined as $|S_c^*|$. By definition, $S_r^* = C$ and $|S_r^*| = n$.

▶ **Lemma 6.** *Given a rooted skeleton S and an integer $1 < w \leq n$, there exists a node c of S such that $w \leq |S_c^*| \leq 3w$.*

Using this, we locate a module $h \in S$ (highlighted with a red marker in Figures 6(a) and 6(b)) such that $|S_h^*| \in \Theta(P)$. Afterwards, we "thicken" S_h^* into a structure called the *exoskeleton.*

Exoskeleton. An exoskeleton is made from three types of modules: *core*, *shell*, and *tail* modules; depicted respectively in dark gray, purple and pink in the figures. Recall that we wish to make the neighborhood of S_h is full. The core modules occupy the positions originally occupied by the skeleton S_h, the shell modules are the ones "coating" the skeleton, and the tail modules are "leftover" modules in the neighborhood of a leaf. We recursively thicken S_h by applying this process to its smaller subtrees, effectively converting tail or core modules at the leaves into a shell module near the root. If the core modules cover the entire subtree of S_h, it may take a linear number of transformations to move a module from a leaf to the root. Instead, we allow empty positions inside the exoskeleton which permits us to "teleport" a module from a leaf to the root in $\mathcal{O}(1)$ transformations. Although some of the positions are empty, the structure remains connected due to its shell and the fact that these empty positions are well separated (in tree metric along S_h, see Property 4 below).

Formally, we define an *exoskeleton* X_c as follows. The *core* $\overline{X_c}$ of X_c are positions in the lattice (not necessarily full) that form a subtree of S_c containing c (since we recursively "shave off" leaves to make material for the shell). Note that $\overline{X_c}$, unlike S_c, is not a subgraph of the configuration's dual graph because of the empty positions. In our figures, we highlight core cells with an internal circle. Let $\mathcal{L}$ be the set of positions corresponding to the leaves of $\overline{X_c}$. We call $N^*(\overline{X_c} \setminus \mathcal{L}) \setminus \mathcal{L}$ the *shell* of X_c (recall $N^*(S)$ is the open neighborhood under vertex-adjacency). The following must hold:

1. All modules in X_c are in the neighborhood of its core. ($|\overline{X_c}| \geq 2$, and $X_c \subset N^*\left[\overline{X_c}\right]$.)
2. The leaves are occupied. ($\mathcal{L} \subset X_c$.)
3. All cells in the shell are occupied. ($N^*(\overline{X_c} \setminus \mathcal{L}) \setminus \mathcal{L} \subset X_c$.)
4. The depth (in $\overline{X_c}$) of every empty cell is congruent to k (mod 4) for a fixed $k \in \{0, 1, 2, 3\}$.

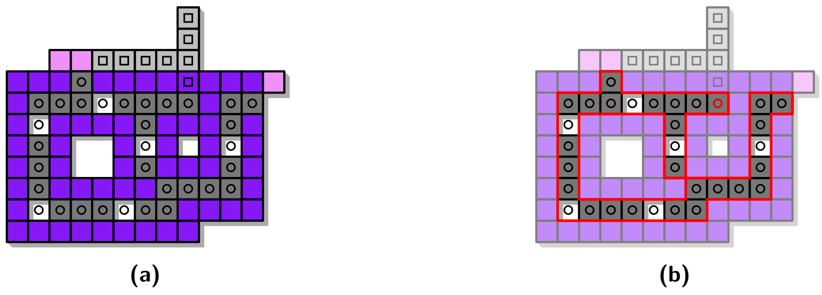


Figure 7 Visualizing an exoskeleton. (a) Skeleton and core cells are highlighted with an inner square and circle, respectively. (b) Highlights the core; the root is highlighted with a red circle.

The modules that are in $N^*(\ell)$, for a leaf $\ell \in \mathcal{L}$, and not in the shell or core of X_c are called the *tail* modules of ℓ. By definition, a module in X_c is either in the shell, in the core, or is a tail module. Shell modules protect the connectivity of X_c, allowing us to place empty positions in the core wherever necessary to expand the exoskeleton constant time.

We show that S_h can be reconfigured into an exoskeleton X_h in $\mathcal{O}(P)$ transformations, thereby inductively establishing Lemma 9. To this end, we first prove Lemma 7, demonstrating how small skeletons (lighter than 9) are turned into exoskeletons. In doing so, we obtain Corollary 8, which settles the base case of our induction. Lemma 7 will also be pivotal in the inductive step. All that this lemma does is take a subskeleton of weight ≤ 9 and transform it into a 3×3 square, a minimal instance of an exoskeleton ($|X_c| = 2$).

▶ **Lemma 7.** *Let C be a configuration with skeleton S in which potentially some of its subskeletons were reconfigured into exoskeletons. Let c be a skeleton node, with parent node d, such that S_c is not yet part of an exoskeleton and $|S_c^*| \leq 9$. Let M be the set of modules in S_c^* that are not contained in exoskeletons. Then, in $\mathcal{O}(1)$ transformations, M can be reconfigured so that:*

- *if* $\deg(d) = 2$, *either $N^*[d]$ is full or $M \cap (N^*[c] \setminus N^*[d])$ is empty (i.e., the modules in M are all contained in the neighborhood of d if this region is not full);*
- *if* $\deg(d) > 2$, *either $N^*[d] \cap N^*[c]$ is full or $M \cap (N^*[c] \setminus N^*[d])$ is empty (i.e., the modules in M are all contained in the intersections of the neighborhood of d and c if this region is not full).*

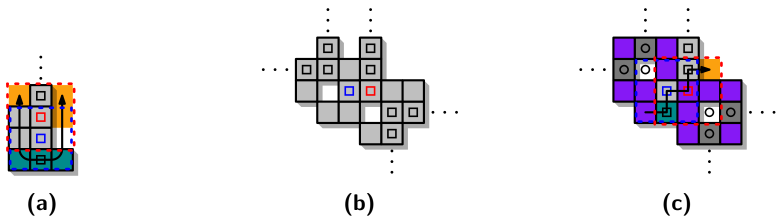


Figure 8 Illustration of Lemma 7. Modules d and c are highlighted with inner red and blue squares. The boundaries of $N^*[d]$ and $N^*[c]$ are shown with dashed red and blue lines.

At first glance, the lemma might seem overly complicated, since without any obstacle, modules can freely move along the perimeter of S (see Figure 8(a)). However, nonlocal pieces of the skeleton (connected subsets S_1 and S_2 of S are *local* if $S \cap N^*(S_1 \cup S_2)$ is connected) previously converted to exoskeletons can interfere when processing later subskeletons. Figures 8(b) and 8(c) show a configuration before and after two nonlocal pieces of the skeleton were converted into exoskeletons. Modules in the working subskeleton S_c might be part of the shell of another exoskeleton, making their movement dangerous (potentially disconnecting the nonlocal exoskeletons). We try to leave such modules in their place, changing their membership from S_c^* to the shell of the exoskeleton.

By applying Lemma 7 a constant number of times in appropriate subskeletons, we eventually obtain a solid 3×3 square from which we can construct an exoskeleton.

▶ **Corollary 8.** *Given a subskeleton S_c with $9 \leq |S_c^*|$, we can reconfigure S_c^* transforming one of its subskeletons into an exoskeleton in $\mathcal{O}(1)$ many transformations, without changing other exoskeletons or modules not in S_c^*.*

Corollary 8 provides the base case of our induction for Lemma 9. We now establish some routines for the inductive step. We assume there is a module $d \in S_h$ so that for every child c of d, S_c^* was reconfigured to an exoskeleton X_c satisfying Property 4 with $k = 0$. If $N^*[d]$ is full, we can finish by making d the root of the exoskeleton, effectively merging the children exoskeletons. Note that this will necessarily happen if d had degree 4 and is not part of a 4-cycle. Thus, assume that there is an empty position $e \in N^*[d]$. We apply a routine Inchworm-Push that fills e with local moves, using a module initially in the core of the exoskeleton X_c (making that cell empty). The routine Inchworm-Pull then moves core modules higher (in tree metric) pushing empty positions deeper until they "exit" the exoskeleton at leaves. Again, ignoring nonlocal interactions, these operations are straightforward. However, in the general case, these interactions mandate extra care.

- Inchworm-Push: Let c be a child of d, and p be the parent of d. Require that $N^*[c]$ is full. We branch into cases. (i) If c, d, and p are collinear, there is a path π in $N^*(d)$ from c to e going through only occupied cells, see Figure 9(a); (ii) If c, d and p form a bend, there is a path π in $N^*[d]$ from c to e going through d and at most one nonskeleton module, see Figures 9(b) and 9(c); (iii) In analogous cases to (i–ii), if a node in $\{c, d, p\}$ is a 4-cycle, there is a path π in $N^*(d)$ from a module in c to e going through one module in the shell of X_d, depicted in Figure 9(d). For all cases, move the modules along π making e occupied and c empty. This requires at most two transformations since there is at most one collision along π and we can decompose π into two paths with no collisions.
- Inchworm-Pull: The operation consists of four stages, each involving at most two transformations. For every empty cell p in the core, select one of its children q arbitrarily, and let $x(q)$ be its x-coordinate. Move the module from q to p in stage $j \in \{1, 2, 3, 4\}$ if

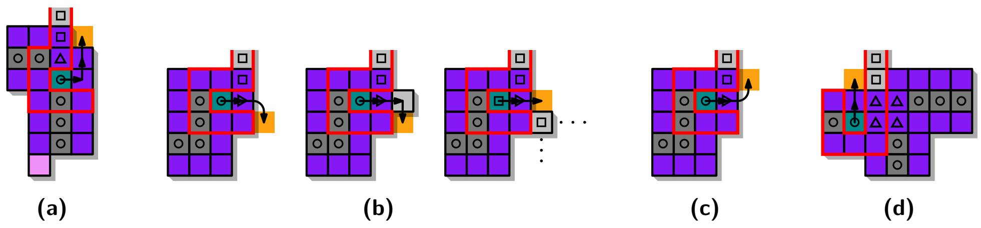


Figure 9 INCHWORM-PUSH. Module c is colored turquoise, cell e is yellow, and d is highlighted with an inner triangle. The red curve encloses a subconfiguration around the moves that remains connected guaranteeing that the transformations do not disconnect the configuration.

$x(q) \equiv j \pmod 4$. Figure 10(a) gives two examples, in one of which p is a 4-cycle (that takes two transformations). If q is a leaf, if there is a tail module of q that is originally in S_c^* and not in another exoskeleton, move such a module to q, visualized in Figures 10(b) and 10(c). If there are no more tail modules we can update $\overline{X_c}$ by deleting q which causes the tail module that just moved to become a shell module, see Figure 10(b), or causes two shell modules to become tail modules as in Figure 10(c).

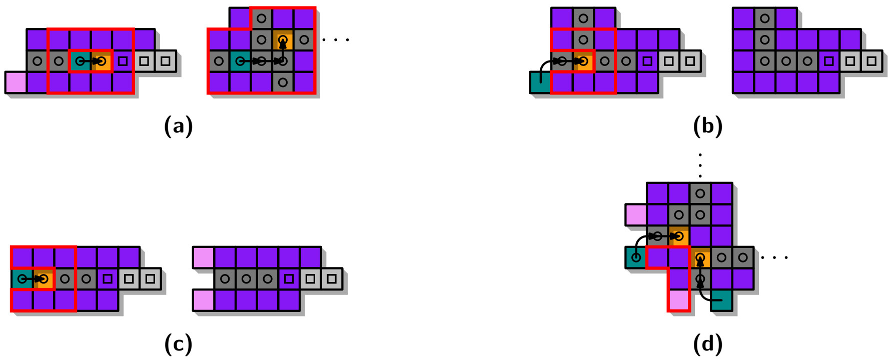


Figure 10 INCHWORM-PULL. Module q is colored turquoise and empty cell p is yellow.

The reason we stagger the moves into four stages in INCHWORM-PULL is to guarantee that the configuration contains enough static modules around a move. Otherwise, performing all moves in a single transformation might disconnect the configuration, as shown in Figure 10(d). In the full version [5], we argue that INCHWORM-PUSH and INCHWORM-PULL preserve connectivity and require only $\mathcal{O}(1)$ transformations. To establish local connectivity of the static modules in the neighborhood of each move, we rely on Properties 3–4.

▶ **Lemma 9.** *Given a configuration C with skeleton S, and a skeleton node d with $|S_d^*| \geq 9$, $\mathcal{O}(|S_d^*|)$ transformations are sufficient to reconfigure S_d into an exoskeleton X_d.*

Proof. If a subskeleton of S_d does not contain an exoskeleton, we can apply Corollary 8. If, after that, S_d is not the core of an exoskeleton X_d, we apply induction. Let Q be the set containing the children of d. For every child $c \in Q$ with $|S_c^*| \geq 9$ we can get an exoskeleton X_c in $\mathcal{O}(|S_c^*|)$ transformations by inductive hypothesis. For every child $c \in Q$ with $|S_c^*| < 9$, we apply Lemma 7 that either results in an exoskeleton X_c, or results in deleting c from S "compacting" all its modules in $N^*[c] \cap N^*[d]$. If after that $N^*[d]$ is not full, $N^*[d]$ contains at most three empty cells if d is a degree-2 bend in S, at most two empty cells if d is a "straight" degree-2 in S, or at most 1 empty cell if d has degree 3. (Note that if d has degree 4 and is not a 4-cycle, then $N^*[d]$ is full.) For each of these empty cells, we can fill them by applying Inchworm-Push followed by at most four applications of Inchworm-Pull to ensure that $N^*[c]$ is full for all children c of d while maintaining Property 4. This takes at most $\mathcal{O}(1)$ transformations. When $N^*[d]$ is full, the configuration contains an exoskeleton X_d except for the "4-arity" of the empty positions (Property 4). That can be resolved by applying Inchworm-Pull to X_c at most three times for each child c of d. ◀

By Lemma 6, we can choose an appropriately heavy node d of S to obtain:

▶ **Corollary 10.** *Let C be a configuration with bounding box perimeter P. We can reconfigure C so that it contains an exoskeleton X_h with $\geq 36P$ modules in $\mathcal{O}(P)$ transformations. All modules stay within $\mathcal{O}(1)$ distance from the bounding box of C.*

4.2 Phase (II): Scaffolding

Phase (I) of our algorithm recursively constructed a sufficiently large exoskeleton X_h that is now reconfigured in **Phase (II)**. The goal of this reconfiguration is to create a $\dashv$ -shaped exoskeleton, with its vertical segment precisely aligned along the right boundary of the bounding box. This vertical edge will serve as the sweep line in **Phase (III)**.

Phase (II) of our approach consists of the following three steps:

1. First "compact" X_h so that its core contains no empty cells.
2. We then choose a rightmost node c in the core of X_h and use Inchworm-Push and -Pull to grow a horizontal path from c rightward, until a 3×3 square is formed outside the bounding box, see Figures 11(a)–11(f).
3. Use Inchworm-Push and -Pull to grow the path upwards until it hits the top of the bounding box, then grow a path downwards until it hits the bottom of the bounding box, see Figures 11(f)–11(h).

We make sure that X_c always contains h so that it maintains connectivity with the subset of S that was not transformed into an exoskeleton. We can show that the connectivity properties of exoskeletons allow us to "go through" obstacles (sections of S that remained untouched in **Phase (I)**), see Figures 11(a)–11(f). That may require some extra moves as shown in Figure 11(b), where an Inchworm-Push would move the cut vertex marked with a red $\times$. In such cases, we sequentially fill in up to two cells in the neighborhood of X_c (yellow in Figure 11(b)), making all modules in π (from the definition of Inchworm-Push) noncut.

4.3 Phase (III): Creating scale

Once **Phase (II)** concludes, we are left with an intermediate configuration that contains a scaffold structure along its east boundary, recall Figure 11(h). In the remaining phase, we use this scaffold to obtain a 3-scaled configuration that can be arbitrarily reconfigured due to Theorem 4. To build squares in an efficient manner, we modify part of the exoskeleton structure created during **Phase (II)**, creating a vertical line of 3×3 meta-modules.

Sweep line. We define a scaffold structure composed of *meta-modules*, each consisting of eight modules occupying the open vertex-neighborhood of a common center cell v, i.e., one in every cell of $N^*(v)$ as shown in Figure 12(a). A *sweep line* $\ell \subset C$ is then a valid subconfiguration formed by the union of h disjoint meta-modules $M_1, \ldots, M_h$ with identical x-coordinates such that ℓ spans the full height of the bounding box of $C \setminus \ell$. Let v_i now refer to the center cell of M_i, such that $y(v_i) < y(v_j)$ exactly if $i < j$, see Figures 12(b) and 12(c).

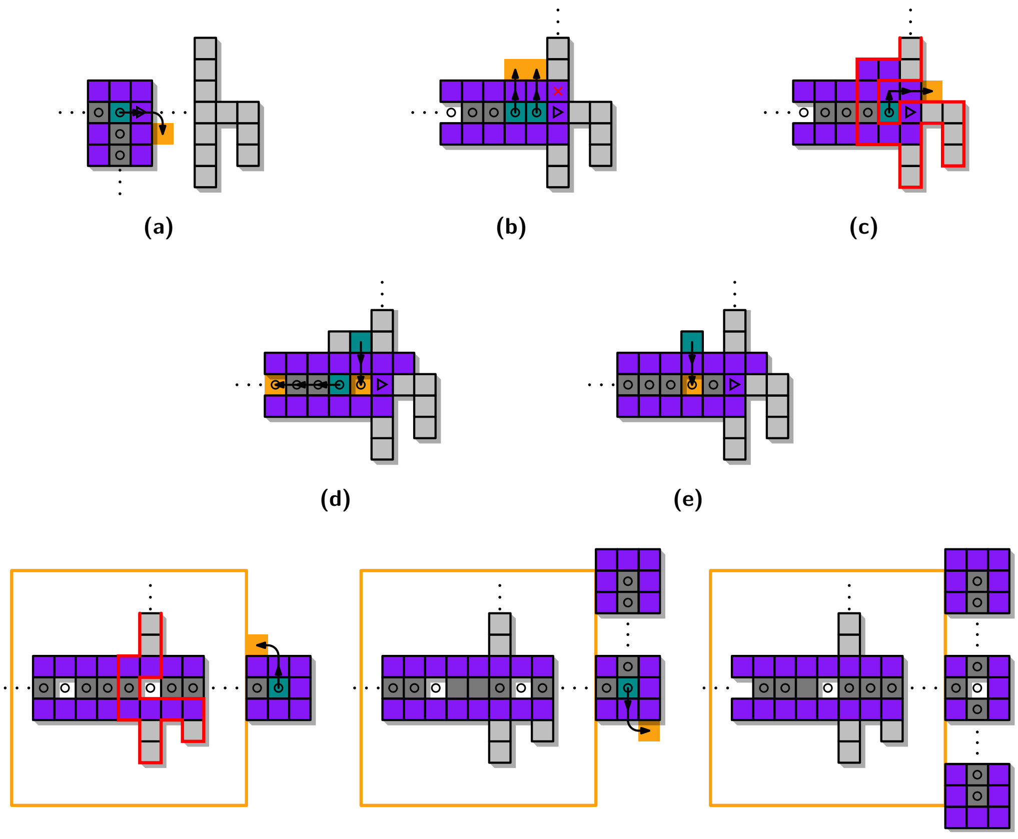


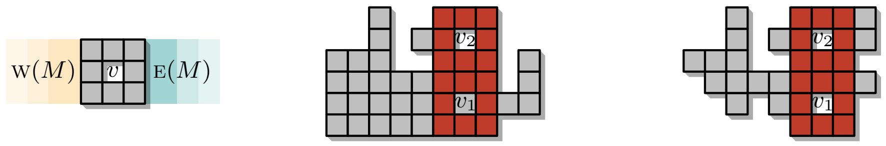
Figure 11 Building the scaffolding; bounding box of the initial configuration is shown in yellow.

For each meta-module M in a sweep line $\ell \subseteq C$, we define as its east and west strips as the cells which share a y-coordinate with one of its modules and are located within the bounding box of the full configuration C. We denote these by $\mathrm{E}(M)$ and $\mathrm{W}(M)$, respectively, and refer to their union for ℓ, e.g., $\mathrm{W}(\ell)$. We use these to define the following characteristics.

A sweep line ℓ is *clean* exactly if every meta-module M_i either has an empty center cell v_i, or its respective west strip is fully occupied, i.e, $\mathrm{W}(M_i) \subseteq C$. See Figure 12(b) for an illustration. Analogously, we call ℓ *solid* if all center cells of its meta-modules are occupied.

Furthermore, we say that is ℓ a *separator* if for every meta-module M_i in ℓ:

1. The east strip $\mathrm{E}(M_i)$ does not contain modules unless all cells in $\mathrm{W}(M_i)$ are full.
2. The modules in $\mathrm{E}(M_i)$ are located in its x-minimal cells, thus forming a 3 cell tall rectangle with at most two additional "loose" modules.



(a) A meta-module M. **(b)** A clean sweep line (red). **(c)** A separator sweep line (red).

Figure 12 Sweep lines are formed by vertical stacks of meta-modules.

We start by transforming part of the exoskeleton from **Phase (II)** into meta-modules that form a sweep line at the eastern boundary of B.

▶ **Lemma 11.** *The vertical portion of the "$\dashv$-shaped" exoskeleton in C can be made into a separator sweep line in $\mathcal{O}(1)$ transformations.*

This gives us a separator sweep line ℓ that is not necessarily clean or solid, located at the east edge of the configuration's bounding box B. Our goal is to move ℓ towards the west edge of B while maintaining its separator property, compacting all encountered modules into 3-tall strips in the process. We define two procedures that suffice to reach this goal.

Intuitively, the Advance operation provides a schedule that translates a sweep line by one unit to the west, and the Clean operation replaces a given sweep line with a clean one. To efficiently parallelize these operations, we split the sweep line into *leading* and *trailing* meta-modules such that M_i is trailing exactly if i is odd, and leading otherwise. At any given time, either the leading or the trailing meta-modules are active, but never both.

By repeatedly alternating between the Advance and Clean operations, we create a configuration with a clean separator sweep line that has an empty west half-space, see Figure 13.

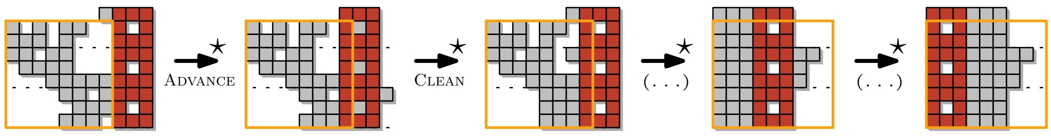


Figure 13 We alternate between the Advance and Clean operations until $\mathrm{W}(\ell) \cap C = \varnothing$.

We first argue the Clean operation can be performed in constantly many transformations. Let M_i be an active meta-module with an occupied center cell. On a high-level, we consider a rectangle induced by occupied cells in west direction within its strip. It is easy to see that we can fill an empty position immediately adjacent to this rectangle, while also cleaning the center cell of M_i by at most two subsequent transformations. By doing this for leading and trailing meta-modules separately, we conclude the following lemma.

▶ **Lemma 12.** *A configuration C with a sweep line ℓ can be cleaned by 4 strictly in-place transformations, without moving any modules east, or exceeding the bounding box B of C.*

It remains to argue that the same is true of the Advance operation. For this, we consider the three positions immediately west of an active meta-module, and distinguish eight cases based on occupied and empty cells. For all of them, we define local transformation schedules that enable us to move a meta-module one step to the east. Thus, we obtain:

▶ **Lemma 13.** *A configuration C with a clean separator sweep line ℓ can be advanced into a clean separator sweep line by $\mathcal{O}(1)$ strictly in-place transformations if $C \cap \mathrm{W}(\ell) \neq \{\varnothing\}$.*

After no more than $\mathcal{O}(P)$ many iterations consisting of Clean+Advance, we obtain a state in which $\mathrm{W}(\ell) \cap C = \varnothing$. The separator property implies that the modules in $\mathrm{E}(\ell)$ form 3-wide, horizontal stacks. It remains to show that we can make this configuration 3-scaled.

To simplify our arguments, assume that n is a multiple of nine. In the general case, n can exceed a multiple of nine by at most 8 modules. We can place these at the south-west corner of the extended bounding box B' and locally ensure the existence of a connected backbone during each transformation.

▶ **Lemma 14.** *Let C contain a clean separator sweep line ℓ with an empty west half-space. Then C can be transformed into a 3-scaled configuration in $\mathcal{O}(P)$ transformations.*

On a high level, we achieve this by filling the center cells of all meta-modules and balancing the remaining modules between their east strips such that each contains a multiple of nine. Having constructed a 3-scaled configuration, we can arbitrarily reconfigure due to Theorem 4 before applying **Phases (I-III)** in reverse to obtain our target configuration.

5 Conclusions and future work

We have presented several new results for reconfiguration in the sliding squares model, making full use of parallel motion. In particular, we showed that deciding whether there exists a single transformation that reconfigures one configuration into another is already NP-complete. Additionally, we provided algorithmic results for the unlabeled variant, including a polynomial-time algorithm that performs in-place reconfiguration in $\mathcal{O}(P)$ transformations, where P denotes the perimeter of the union of the configurations' bounding boxes.

We note that this algorithm can easily be adapted to the labeled setting by "sorting" the modules in the xy-monotone configuration using $\mathcal{O}(P)$ transformations. However, this requires a relaxation of the in-place requirement. Furthermore, deciding whether a schedule of makespan 1 exists in the labeled setting is straightforward, while the problem becomes NP-complete for a makespan of 2. Details on these results are provided in the full version [5].

Our algorithmic results are only worst-case optimal, and there are a number of possible generalizations and extensions of the setting. Previous work in the sequential setting 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.

Another aspect is a refinement of the involved parameters, along with fixed parameter tractability. Our hardness proofs show that the investigated problems are not FPT when parameterized by the number of transformations in the output. The number of modules in the input configuration is not a suitable parameter since it is the input size. Are there other parameters for which these problems are FPT? An interesting candidate would be the size of the symmetric difference between the two input configurations. In our hardness proofs, the size of the symmetric difference is linear on the size of the SAT instance.

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, 2025. `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 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated motion planning: The video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. `doi:10.4230/LIPICS.SOCG.2018.74`.

8 Julien Bourgeois, Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Benoît Piranda, Christian Rieck, and Christian Scheffer. Space ants: Episode II - Coordinating connected catoms. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:6, 2022. `doi:10.4230/LIPIcs.SoCG.2022.65`.

9 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`.

10 Matthew Connor, Othon Michail, and George Skretas. All for one and one for all: An $\mathcal{O}(1)$-musketeers universal transformation for rotating robots. In *Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 9:1–9:20, 2024. `doi:10.4230/LIPIcs.SAND.2024.9`.

11 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`.

12 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. `doi:10.1137/18M1194341`.

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

14 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for fast locomotion of metamorphic robotic systems. *International Journal of Robotics Research*, 23(6):583–593, 2004. `doi:10.1177/0278364904039652`.

15 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`.

16 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`.

17 Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Jonas Neutzner, Christian Rieck, and Christian Scheffer. Coordinated motion planning: Multi-agent path finding in a densely packed, bounded domain. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 29:1–29:15, 2024. `doi:10.4230/LIPIcs.ISAAC.2024.29`.

18 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`.

19 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`.

20 Toshio Fukuda, Seiya Nakagawa, Yoshio Kawauchi, and Martin Buss. Structure decision method for self organising robots based on cell structures-cebot. In *International Conference on Robotics and Automation (ICRA)*, pages 695–696, 1989. `doi:10.1109/ROBOT.1989.100066`.

21 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.

22 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.

23 Keith D. Kotay and Daniela L. Rus. Algorithms for self-reconfiguring molecule motion planning. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2184–2193, 2000. doi:10.1109/IROS.2000.895294.

24 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019. doi:10.1016/j.jcss.2018.12.001.

25 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.

26 An Nguyen, Leonidas J. Guibas, and Mark Yim. Controlled module density helps reconfiguration planning. *New Directions in Algorithmic and Computational Robotics*, pages 23–36, 2001.

27 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.

28 Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10:107–124, 2001. doi:10.1023/A:1026504804984.

29 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.

30 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.

31 Matthijs Wolters. Parallel algorithms for sliding squares. Master's thesis, Utrecht University, 2024.

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