Input-Sensitive Reconfiguration of Sliding Cubes

Hugo A. Akitaya* Matias Korman† Frederick Stock*

Abstract

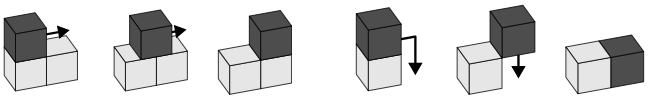
A configuration of n unit-cube-shaped *modules* (or *robots*) is a lattice-aligned placement of the n modules so that their union is face-connected. The reconfiguration problem aims at finding a sequence of moves that reconfigures the modules from one given configuration to another. The sliding cube model (in which modules are allowed to slide over the face or edge of neighboring modules) is one of the most studied theoretical models for modular robots.

In the sliding cubes model we can reconfigure between any two shapes in $O(n^2)$ moves ([Abel *et al.* SoCG 2024]). If we are interested in a reconfiguration algorithm into a *compact configuration*, the number of moves can be reduced to the sum of coordinates of the input configuration (a number that ranges from $\Omega(n^{4/3})$ to $O(n^2)$, [Kostitsyna *et al.* SWAT 2024]). We introduce a new algorithm that combines both universal reconfiguration and an input-sensitive bound on the sum of coordinates of both configurations, with additional advantages, such as $O(1)$ amortized computation per move.

1 Introduction

Modular self-reconfigurable robotic systems offer several advantages over typical robotic systems. Such systems are composed of a set of robotic units (called **modules**) that can communicate, attach, detach, and move relative to each other. This means modules can move to change the overall system's shape, providing versatility to perform unforeseen tasks. While advantageous, the increased degree of freedom also incurs an algorithmic challenge, as it is not easy to determine how to use the module's operations so that the union of all modules creates a specific shape (this is known as the **reconfiguration** problem). The algorithmic community has investigated the reconfiguration problem for many variations of modular robots, be it the type of moves allowed (sliding [5, 9, 10], pivoting [3, 13], and so-called square "atoms" [7, 8]) or shape (such as hexagons [4]).

We investigate one of the most established models, which is called the **sliding (hyper-)cube model**:



(a) (b)

Figure 1: (a) Slide and (b) Convex transition.

modules are lattice-aligned unit (hyper-)cubes where two modules are **adjacent** if they share a face (i.e., a $(d-1)$-dimensional facet). In this model, two types of moves are allowed: a module can slide along the co-(hyper-)planar faces of two other adjacent modules, or it can slide along two orthogonal faces of another module (see Fig. 1).

The **reachability** problem asks whether, given two configurations with n modules, there exists a sequence of moves that can transform one into the other. If the answer is always 'yes' we say that the model admits **universal** reconfiguration.

The first universal result for the two dimensional sliding model was obtained by Dumitrescu and Pach [9] with a reconfiguration algorithm that required $O(n^2)$ moves. This result could be seen as optimal since some instances require $\Omega(n^2)$ moves. The 2-dimensional result was afterwards extended to three and higher dimensions in an informal publication of Abel and Kominers [2]. Their paper claimed universal reconfiguration and that $O(n^3)$ many moves were always sufficient.

Other research [3] focused on variations of the model, in both the shape of the modules and the type of movements that were allowed. They obtained characterization of which variations of the model have universal reconfiguration. For the versions in which no universal reconfiguration algorithm exists, they showed that determining if a particular instance can be reconfigured into a target configuration is PSPACE-complete.

Recently there have been two simultaneous, independent results in three dimensions. Abel *et al.* [1] formally published their preliminary result, reducing the number of moves to $O(n^2)$. In the same paper, the authors explain how the algorithm can be made in-place and input-sensitive (the exact bound depends on the dimension of the bounding box of both configurations). **In-place** means throughout the reconfiguration there are never more than a constant number of modules outside the bounding boxes of the start and end configuration. Further, these modules are never more than

*University of Massachusetts Lowell, Lowell, USA, {hugo_akitaya@, frederick_stock@student.}uml.edu.
Supported by NSF grant CCF-2348067.
†Siemens Electronic Design Automation, Wilsonville, USA, matias.korman@siemens.com

distance 1 from the bounding box [1]. Parallel to [1], Kostitsyna *et al.* [12] independently produced a different input-sensitive reconfiguration algorithm: a configuration C can be reconfigured into a **compact configuration**, intuitively, a shape where all modules are clumped together without holes (formal definition given in Section 2). Their algorithm takes $O(\sum_{\mathbf{m}\in C_1}\|\mathbf{m}\|_1)$ sliding moves where $\|\mathbf{m}\|_1$ denotes the L_1 norm of the position of a module $\mathbf{m}$. This result has a better input-sensitive bound, but is slightly limited: they only show how to reconfigure into a compact configuration. Since there is more than one compact configuration with the same number of modules, this does not directly lead to a universal reconfiguration strategy. Reconfiguration between compact configurations is significantly simpler than compactification, but does not immediately follow.

This paper we improve on both [1] and [12]: we give a universal reconfiguration whose number total number of moves depends on the sum of input and target coordinates ($O(\sum_{\mathbf{m}\in C_1}\|\mathbf{m}\|_1 + \sum_{\mathbf{m}\in C_2}\|\mathbf{m}\|_1)$. Since configurations are connected, it is easy to see that this bound is in $\Omega(n^{4/3}) \cap O(n^2)$ (when all modules form a solid cube or when they form the 1-skeleton of a larger cube, respectively). Our main result is formally stated as follows.

Theorem 1 *Given two configurations C_1 and C_2 with n cube modules in 3 dimensions where all modules lie in the positive xyz orthant, there is an in-place reconfiguration between them that uses at most $18(\sum_{\mathbf{m}\in C_1}\|\mathbf{m}\|_1 + \sum_{\mathbf{m}\in C_2}\|\mathbf{m}\|_1) + 120n + O(1)$ sliding moves where $\|\mathbf{m}\|_1$ denotes the L_1 norm of the position of a module $\mathbf{m}$. Such a reconfiguration can be computed in $O(1)$ amortized time per sliding move.*

In our algorithm we use a constant number of auxiliary "helper" modules that will allow nearby modules to move. We follow existing literature [3] and refer to the auxiliary modules as **musketeers**. The multiplicative factor 120 may seem large, we note that our algorithm uses six musketeer modules, so this constant is better characterized by $20n$ moves per musketeer module. Further, since we reconfigure between C_1 and C_2 by reaching an intermediate configuration, this means that we actually need $10n$ moves per musketeer to transform any configuration of n modules into *compact* form.

Although the main emphasis is in the total number of moves required, it is also interesting to bound the time needed to compute the sequence of moves needed to reconfigure between two configurations. Our algorithm can compute each move in $O(1)$ amortized time. Although the computation time is not directly addressed in the previous papers, it is not hard to see that the algorithm of [1] also achieves $O(1)$ amortized (after a BFS traversal on both configurations, each step is easily computed in O(1) time). On the other hand, a naïve implementation of [12] would require $\Theta(n)$ time per move as a global search must be executed each time. Due to space constraints some proofs are omitted. Details can be found in the extended version of this paper [6]

[1] Traditionally, in-place algorithms are a bit more restrictive, only one module is allowed outside the bounding box, making both constants exactly 1. In this paper we relax the condition for ease of description.

2 Definitions and Preliminaries

A **configuration** C of n unit cube modules is a set of n cells in the cube lattice. A cell is **occupied** if it is in C or **empty** otherwise. We abuse notation by sometimes referring to occupied cells as modules. Two cells are **adjacent** if they share a face. Let the **adjacency graph** G_C be the graph whose vertices are the cells in C and edges are defined by pairs of adjacent occupied cells. We call C **connected** if G_C is connected. A module is **articulate** if it is a cut vertex in G_C, and **nonarticulate** otherwise. For an occupied cell m, the notation $\mathbf{m}$ refers to the module at cell m. Note that a connected configuration C defines a polycube. We denote by ∂C the boundary of C. The **outer boundary** of C is the boundary of the unbounded component of the complement $\overline{C}$ of C. We say a module is "in the outer boundary" if at least one of its faces lies in the outer boundary of C.

A **move** is an operation that transforms an n-module configuration C into another C' so that $C \cap C'$ is a configuration of $n-1$ modules (and thus $C \setminus C'$ is a set with one module). We require the **single backbone condition**: a move between connected configurations C and C' is only allowed if $C \cap C'$ is also connected. We say that the module in position $C \setminus C'$ **moved** to position $C' \setminus C$. In other words, the single-backbone condition requires each moving module to be nonarticulate.

The **sliding model** allows two types of moves (refer to Fig. 1):

- A **slide** moves a module $\mathbf{a}$ from 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 **convex transition** moves a module $\mathbf{a}$ from 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.

Note a slide requires the final cell b to be empty, and a convex transition requires the target b as well as an intermediate cell to be empty. These are called the **free-space requirements** of the moves. If these requirements are not met, performing either of these moves

would cause two modules to collide. Note that a module might be nonarticulate but is not allowed to move if the free-space requirements mentioned above are not satisfied. We call a module **movable** if it is nonarticulate and satisfies the free-space requirements of either a slide or a convex transition.

We may refer to a cell using the coordinates of its closest corner to the origin. Hence cell $(0,0,0)$ corresponds to the unit cube with vertices $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(1,0,0)$, $(0,1,1)$, $(1,1,0)$ and $(1,1,1)$. We denote by $\|\mathbf{m}\|_1$ the L_1 norm of the cell occupied by a module $\mathbf{m}$. For example, if $\mathbf{m}$ occupies $(x_\mathbf{m}, y_\mathbf{m}, z_\mathbf{m})$, then $\|\mathbf{m}\|_1 = |x_\mathbf{m}| + |y_\mathbf{m}| + |z_\mathbf{m}|$.

We assume that both configurations are contained in the positive orthant and that a corner of the bounding box of either configuration is the origin.

Our algorithm uses the **slice graph** from Fitch and Rus [11]. For a connected configuration C of n modules and $z_0 \in \mathbb{Z}$, the **slice** at height z is $C \cap \{z = z_0\}$ (the modules whose z-coordinate is equal to z_0). Each maximally connected component in a slice is called a **cluster**. Each cluster defines a vertex of the slice graph. Two vertices are connected if at least one module in each cluster share a face (see Fig. 2).

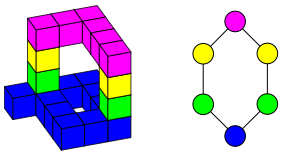


Figure 2: A configuration and its slice graph.

A cell c **dominates** another cell c' if c' is contained in the minimum axis aligned bounding box containing c and the origin. We also say that $c = (x, y, z)$ is **below** $c' = (x', y', z')$ if $x = x'$, $y = y'$ and $z < z'$ (similarly, we define **above** in a similar way. A module is called **compact** if every cell dominated by it is occupied. A module is called **quasi-compact** if every cell that is dominated by it and is not below it is occupied. A configuration or cluster is **compact** (resp., **quasi-compact**) if every module in it is compact (resp., quasi-compact). A cluster is **extremal** if not all of its modules are quasi-compact, and all adjacent modules above it (if any) are quasi-compact. (Note that a cluster above a non-quasi-compact cluster may be quasi-compact.)

2.1 General strategy.

A common technique in reconfiguration algorithms is to define a **canonical** configuration C^* and describe a sequence of moves that transform a given configuration C into C^*. Because the moves are reversible, this implies a solution for the reachability problem: Given two configurations C and C', one can obtain reconfiguration sequences from both to C^*, and then compose the reconfiguration from C to C^* with the reverse of the reconfiguration from C' to C^*. Instead of a single configuration, we use a class of canonical configurations: all compact configurations of n modules. The strategy is to compute a sequence of moves to transform C and C' to two configurations D and D' of this class. We then transform C into D, then D into D', and finally reconfigure D' to C'. Thus, our algorithm can be divided into two phases: reconfiguration between a given configuration and a compact one (Section 3); and reconfiguration between two compact configurations (Section 5).

3 Compacting configurations

In this section, we describe an algorithm we call CompactIfy, which reconfigures a given configuration into a compact one. As a preprocessing step, use LocateAndFree (from [1]) to obtain six movable modules on the outer boundary. These modules will be referred to as **musketeers**.

Once done, the algorithm repeatedly picks an extremal cluster S and uses the musketeers to move every module in S without a lower neighbor to a position with a smaller z-coordinate. By doing so, S will merge with at least one cluster in the slice below. Initially, a module is only moved if the position below it is empty. Otherwise, it is left where it was. We call these leftover modules **stragglers**. We use another procedure Fix, to find a suitable position for the stragglers. The result is that every module previously in S either has a lower z-coordinate or becomes quasi-compact. By iterating over all clusters we will finish with a compact configuration or all modules in the $z = 0$ plane. Pseudocode of CompactIfy is presented in Alg. 1 (in Section A).

3.1 Obtaining Musketeers

The main role of the musketeers will be to provide temporary connectivity to nearby modules. We use a subroutine LocateAndFree from [1] to gather them above the highest cluster.

Lemma 2 *A constant number k of musketeer modules can be positioned on the outer boundary of any configuration in at most $4kn + O(1)$ moves.*

3.2 Lowering Modules

Given an extremal cluster S let S^* be the set of non-quasi-compact modules in S, and let T be a minimum Steiner tree of S^* using the quasi-compact modules in S (if any) as Steiner vertices. We process T in post-order, rooted arbitrarily, moving it to the z-layer below if it has no lower neighbor. Lowering is accomplished via the musketeer modules (shown in blue in Figs. 3 and 4).

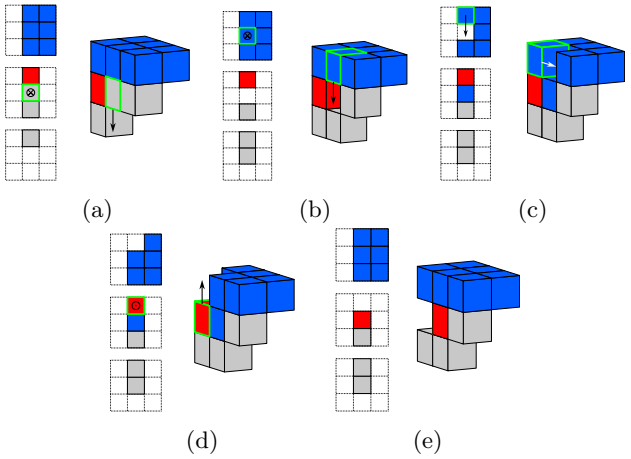


Figure 3: The four moves used to lower a straight degree-2 module in an extremal cluster using the musketeers for connectivity. The left of each figure shows the top view of the three relevant z-layers. $\odot$ and $\otimes$ represent arrows going into and out of the plane, respectively. The moving module is highlighted in green.

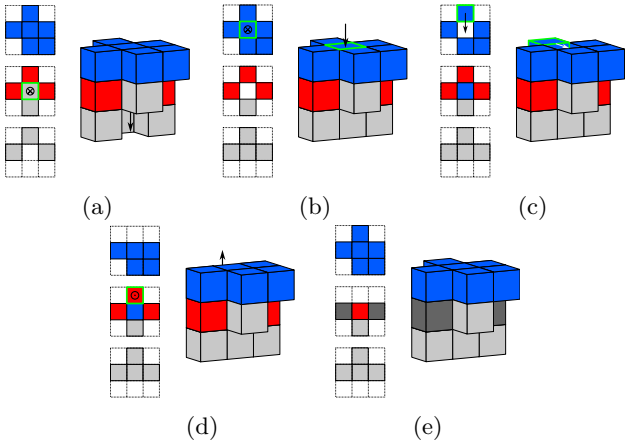


Figure 4: Four moves used to lower a degree-4 module.

In red we show auxiliary modules that stablish the connectivity between the processed and unprocessed parts of T. If $\mathbf{p}$ has a lower neighbor, it cannot move directly down, thus it becomes a straggler. (After completing our traversal of T we return and "fix" them.)

Since we visit modules in post order, we can guarantee the children of the module currently being processed (if any) have been already processed. This implies that, when processing a module, the neighbors in the same z-layer are either red or is the parent of the module in T. The exact strategy depends on the degree of $\mathbf{p}$ in T. The strategies for a straight degree-2 (child and parent share either x or y coordinate) and degree-4 is shown in Figs. 3 and 4, respectively. Degree-3 and bent degree-2 modules are handled similarly as degree-4. Details for other cases are in the full version [6].

Lemma 3 *Given a Steiner tree T of the non-quasi-compact modules of an extremal cluster S and 6 musketeer modules positioned above T as in Fig. 3a or Fig. 4a, let ℓ be the number of modules that were lowered and s be the number of remaining stragglers ($|T| = \ell + s$). Then, $17\ell + 13s$ moves suffice to move down by one unit every module in T that has no lower neighbor while maintaining connectivity.*

Proof sketch. Each module is visited on average 2 times by the musketeers: for every high-degree module in T (degree 3 or 4) there is a leaf that does not need to be visited. Each visit costs 6 moves totaling 12 moves per module. Compactify moves stragglers toward the leaves of T and each module is the base of a straggler at most once (13 moves per module). The lowered modules cost 4 extra moves depicted in Fig. 3 or Fig. 4. □

3.3 Handling straggler modules

After completing our post-order traversal, there is at least one straggler $\mathbf{m}$. Since $\mathbf{m}$ is not quasi-compact, there is at least one empty position in the cuboid region dominated by $\mathbf{m}$ and not directly below it. In Fix($\mathbf{m}$) we search for such an empty position p and effectively move $\mathbf{m}$ to p, either directly or through a chain of moves that fill p and leave m empty. The pseudo-code for Fix is presented in Alg 2 (in the Appendix) .

For a given cell $p = (x_p, y_p, z_p)$, we denote by $\square(p)$ the set of cells $(x_p - i, y_p - j, z_p)$ for all $0 \le i \le x_p$ and $0 \le j \le y_p$, i.e., the cells intersecting the minimum axis aligned horizontal square containing $(0, 0, z_p)$ and p. We define that a straggler is **fixable** if every *module* in $\square(\mathbf{m}) \setminus \{\mathbf{m}\}$ in the same cluster of $\mathbf{m}$ is quasi-compact. We first try moving $\mathbf{m}$ closer to the origin in $\square(\mathbf{m})$ (Fig. 5a). If we can't, either (i) $\mathbf{m}$ moves to a z-layer below it or becomes quasi-compact (and we are done); (ii) $\mathbf{m}$ is on the edge of the bounding box (and we can move through the plane $x = -1$ or $y = -1$ to an empty position); or (iii) $\mathbf{m}$ is blocked by other modules in the same layer that are quasi-compact (as in Fig. 5). Since $\mathbf{m}$ is fixable and non-quasi-compact, there is an empty position p dominated by $\mathbf{m}$ in column $(x_\mathbf{m} - 1, y_\mathbf{m}, .)$ or $(x_\mathbf{m}, y_\mathbf{m} - 1, .)$, adjacent to a module $\mathbf{q}$ in column $(x_\mathbf{m} - 1, y_\mathbf{m} - 1, .)$ (Fig. 5a). A sequence of moves fills p and frees $(x_\mathbf{m} - 1, y_\mathbf{m} - 1, z_\mathbf{m})$ (Fig. 5b). We can then fill that empty position while freeing $\mathbf{m}$'s cell (Fig. 5c).

Lemma 4 *If $\mathbf{m}$ is fixable, Fix($\mathbf{m}$) does not disconnect the configuration C, and performs at most $\|\mathbf{m} - p\|_1 + 2$ moves. After Fix($\mathbf{m}$), if $\mathbf{m}$ remained in the same z-layer, $\mathbf{m}$ is now quasi-compact.*

Proof. Since $\mathbf{m}$ is a straggler, if it is the only module moved by Fix($\mathbf{m}$), it is clear that connectivity is preserved. Lines 1–6 move $\mathbf{m}$ monotonically towards the origin, except potentially for the move that brings $\mathbf{m}$

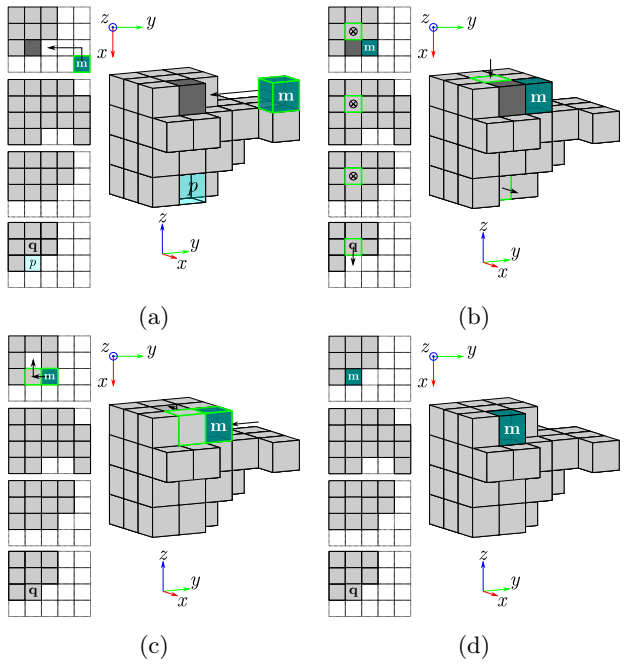


Figure 5: Illustration of the moves in Fix (a) line 2, (b) lines 9–11, and (c) line 12. (d) After Fix, $\mathbf{m}$ becomes quasi-compact.

outside of the bounding box (line 6). The bound on the moves holds in this case, and if $\mathbf{m}$ moved down (in z) the claim holds.

It remains to prove the claim for case (iii), i.e., lines 8–12. By the fact that $\mathbf{m}$ is fixable, when we reach line 8, cells $\mathbf{m}+(-1,0,0)$, $\mathbf{m}+(0,-1,0)$ and $\mathbf{m}+(-1,-1,0)$ must be filled by quasi-compact modules. This implies that all cells with nonnegative coordinates strictly smaller than $\mathbf{m}$'s are full. Since $\mathbf{m}$ is not yet quasi-compact, there must exist at least one empty position dominated by $\mathbf{m}$ and not directly below it. Indeed, all such empty cells must be in columns $(x_{\mathbf{m}}-1, y_{\mathbf{m}}, .)$ or $(x_{\mathbf{m}}, y_{\mathbf{m}}-1, .)$, otherwise $\mathbf{m}+(-1,0,0)$ or $\mathbf{m}+(0,-1,0)$ would not be quasi-compact. Then, p exists and $\mathbf{q}$ is a compact nonarticulate module. Thus, line 9 maintains connectivity. Note that by the choice of p, every cell edge-adjacent in the same z-layer to a module moved by line 11 is full except for the cell in the column $x_{\mathbf{m}}$. Thus, the neighborhood of the moving module remains connected and line 11 maintains connectivity. When we reach line 12, the cell at $(x_p, y_p, z_{\mathbf{m}}-1)$ is full (by the choice of p). The fact that $\mathbf{m}$ is a straggler implies that every edge-adjacent cell above $\mathbf{m}$ have been already processed by Fix, so they are quasi-compact and thus do not require $\mathbf{m}$ or $(x_p, y_p, z_{\mathbf{m}})$ for connectivity. Then, both moves in line 12 maintain connectivity. Except for line 9 and the move from $(x_p, y_p, z_{\mathbf{m}})$ to $\mathbf{m}+(-1,-1,0)$ (line 12), all the moves are monotonic in the direction $p-\mathbf{m}$. Thus the total number of moves is as claimed. □

3.4 Compactify for planar configurations

With a few small adjustments, our algorithm can be applied to planar configurations (when all modules lie in the $z=0$ plane). Note that we allow using planes $z=1$ and $z=-1$ to move musketeers and stragglers. The clusters are now defined by the y-slice graph (maximal components induced by a fixed y coordinate). Lowering modules follows the same principle, except that we lower the y-coordinates of the modules (cases are simpler, as modules have degree 2 or less within the cluster). Fix also becomes substantially simpler as stragglers can simply move to the closest dominated empty cell through the $z=-1$ plane.

4 Analysis of Compactify

Lemma 5 Compactify(C) *transforms a configuration C into a compact configuration maintaining connectivity.*

Proof sketch. The claim is mostly established by Theorem 3 and lemma 4. We process the stragglers in increasing order of L_1 norm, guaranteeing that the first straggler is fixable. By Lemma 4, the straggler either moves down or becomes quasi-compact, making the subsequent processed straggler fixable. □

Lemma 6 *A sequence of $17\sum_{\mathbf{m}\in C}\|\mathbf{m}\|_1+60n+O(1)$ moves suffices to transform a configuration C with n modules into a compact one.*

Proof sketch. By Lemma 3, we can charge 17 moves to the decrease of the z-coordinate of the lowered modules. By Lemma 2 2 we can obtain six musketeers in $24n+O(1)$ moves. While traversing C, every musketeer walks on the bottom face of each module of a cluster at most once, and walks on the side face of one module exactly once (see Fig. 6 for example). This amounts $6n+O(1)$ musketeer movements (each cluster is only traversed in this way once). In the planar version of Compactify the musketeers can move in the positive y-direction using the bottom faces, which adds $6n$ moves. The total amounts $12n+O(1)$ musketeer movements. The musketeers may walk on the top face of a module at most two times, $24n$ moves. Overall, the musketeers preform $24n+12n+O(1)=36n+O(1)$ moves. Combined with the bound from Lemma 2, we get a total $24n+36n+O(1)=60n+O(1)$ moves. □

Lemma 7 Compactify *has runtime that is within an amortized $O(1)$ factor of the moves performed.*

Proof. Whenever a musketeer moves, it is either following a constant size schedule of moves such as Fig. 3, or is traveling along a shortest path from one cluster to another. The former case clearly only requires $O(1)$

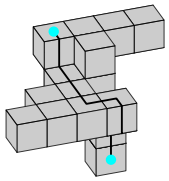


Figure 6: The path of an upwards traversal of musketeers from one cluster to another. Their start and end positions marked by blue circles.

computation, and in the latter, computing the path requires time proportional to the length of the path, hence amortized $O(1)$ computation per movement. The only section of this algorithm where an $O(1)$ computation cost is not simple is Fix, when we must locate an appropriate empty position p. We show if each module stores a constant amount of information, these positions can be located efficiently. Assume that initially each module $\mathbf{m}$ has knowledge of the following three properties: 1) Is $\mathbf{m}$ compact? 2) Is $\mathbf{m}$ quasi-compact? 3) Is every position of $\square(\mathbf{m})$ filled?

Note, each of these are transitive, and, to answer these questions, $\mathbf{m}$ need only query its neighbors with smaller sum of coordinates. Consequently, if $\mathbf{m}$ moves, recomputing if $\square(\mathbf{m})$ is full takes only $O(1)$ time. If $\mathbf{m}$ becomes (quasi-)compact it may be the case that many modules also become (quasi-)compact. While this may affect many more modules than the number of moves performed, a module can become (quasi-)compact only once. Therefore, in aggregate these updates will only take $O(n)$ time, which can then be amortized over the number of moves to $O(1)$ time.

If $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$ (case(ii)), Fix attempts to move it to an empty dominated position p. If $\mathbf{m}$ is not compact, then p exists. Therefore we can determine if p exists in $O(1)$ time. If $\mathbf{m}$ is not compact, we can locate p with $||\mathbf{m} - p||_1$ queries: Checking positions directly below $\mathbf{m}$ (from $\mathbf{m}$) we will either find an empty position (and we are done) or a compact module by the fact that $\mathbf{m}$ is fixable. We have found the z coordinate of p and we can find it by looking in the $-x$ or $-y$ direction (note that $\mathbf{m}$ and p lie in the same plane $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$).

In case (iii), since we know p is either in column $(x_{\mathbf{m}} - 1, y_{\mathbf{m}}, .)$ or $(x_{\mathbf{m}}, y_{\mathbf{m}} - 1, .)$, we can find it with $||\mathbf{m} - p||_1$ queries with a linear search alternating between the two columns. □

5 Reconfiguring Between Compact Configurations

Lemma 8 *Given two compact configurations C_1 and C_2 with n modules where all modules lie in the positive xyz orthant, we can reconfigure one into the other by using at most $\sum_{\mathbf{m}\in C_1} \|\mathbf{m}\|_1 + \sum_{\mathbf{m}\in C_2} \|\mathbf{m}\|_1$ sliding moves.*

Proof. We use induction on $|C_1 \setminus C_2|$. The base case is when $|C_1 \setminus C_2| = 0$ and thus $C_1 = C_2$ and we are done. Otherwise, we claim that there is a module $\mathbf{m}_1 \in C_1 \setminus C_2$ whose cell is not dominated by any other module in $C_1 \cup C_2$. We prove this by contradiction: assume that every module if $C_1 \setminus C_2$ is dominated by some module in $C_1 \cup C_2$. Because domination establishes a partial order, there must exist a module $\mathbf{m} \in C_1 \setminus C_2$ that us dominated by a module in $\mathbf{m}' \in C_1 \cap C_2$. That is, we have found a module $\mathbf{m}$ is only present in C_1 and is dominated by $\mathbf{m}' \in C_1 \cap C_2$ that is present in both C_1 and C_2. However, this is a contradiction to the fact that C_2 is compact. The interesting property of $\mathbf{m}_1$ is that $C_1 \setminus \{\mathbf{m}_1\}$ is compact.

Using a symmetric argument, we show that there exists a cell $\mathbf{m}_2 \in C_2 \setminus C_1$ so that every dominated cell is occupied in $C_1 \cap C_2$. For contradiction, assume no such module exists. Let $\mathbf{m}$ be the module in $C_2 \setminus C_1$ with smallest $\|\mathbf{m}\|_1$. Since C_2 is compact, all cells that $\mathbf{m}$ dominates are occupied in C_2. Let $m' \in C_2 \setminus C_1$ be a module dominated by $\mathbf{m}$, which must exist by assumption. Then $\|\mathbf{m}'\|_1 < \|\mathbf{m}\|_1$, contradicting the choice of $\mathbf{m}$. We conclude that $C_1 \cup \{\mathbf{m}_2\}$ is compact. Furthermore, the two claims combined imply that $C_1' = C_1 \cup \{\mathbf{m}_2\} \setminus \{\mathbf{m}_1\}$ is compact. Then $|C_1 \setminus C_2| = |C_1' \setminus C_2| + 2$ as desired.

To complete the proof we must show that the reconfiguration can be done in the claimed number of moves. Note that $\mathbf{m}_1$ and $\mathbf{m}_2$ must both be in the outer boundary of C_1 and C_1' respectively. By definition of compact, the boundary of $C_1 \setminus \{\mathbf{m}_1\}$ that is not on the $x = 0, y = 0$ or $z = 0$ planes is an x-, y-, and z-monotone surface. Thus, the shortest path from $\mathbf{m}_1$ to $\mathbf{m}_2$ on this surface is also monotone and, hence its length is upper bounded by $\|\mathbf{m}_1\|_1 + \|\mathbf{m}_2\|_1$. □

6 Conclusion

Algorithmic bounds have been derived from the number of modules [2, 9], to dimensions of bounding boxes [1, 5] and eventually the sum of coordinates (this paper and [12]). We note that, none are optimal if the starting and target configurations are far from compact but very similar. Is there a better parameter to bound the length of reconfiguration between configurations? Such a bound is also important for approximation algorithms, of which there are none. Note that in general, shortest reconfiguration is known to be NP-complete [5].

Another avenue of research would be parallelization where we look into the **makespan** (i.e., time required to execute all moves when we allow parallel moves).

References

[1] Z. Abel, H. A. Akitaya, S. D. Kominers, M. Korman, and F. Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in a quadratic number of moves. In *SoCG*, pages 1:1–1:14, 2024.

[2] Z. Abel and S. D. Kominers. Universal reconfiguration of (hyper-)cubic robots. *arXiv preprint*, 2008.

[3] H. A. Akitaya, E. M. Arkin, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, M. Korman, B. Palop, I. Parada, A. v. R. Renssen, and V. Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ Musketeers. *Algorithmica*, 83:1316–1351, 2021.

[4] H. A. Akitaya, E. D. Demaine, A. Gonczi, D. H. Hendrickson, A. Hesterberg, M. Korman, O. Korten, J. Lynch, I. Parada, and V. Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *SoCG*, volume 189, pages 10:1–10:20, 2021.

[5] H. A. Akitaya, E. D. Demaine, M. Korman, I. Kostitsyna, I. Parada, W. Sonke, B. Speckmann, R. Uehara, and J. Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *SWAT*, pages 4:1–4:19, 2022.

[6] H. A. Akitaya, M. Korman, and F. Stock. Input-sensitive reconfiguration of sliding cubes. *arXiv preprint*, 2025.

[7] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, and S. Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry*, 42(6-7):652–663, 2009.

[8] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer. Reconfiguration of cube-style modular robots using $o(\log n)$ parallel moves. In *ISAAC*, pages 342–353. Springer, 2008.

[9] A. Dumitrescu and J. Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006.

[10] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20(3):409–418, 2004.

[11] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *IROS*, volume 3, pages 2460–2467. IEEE, 2003.

[12] I. Kostitsyna, T. Ophelders, I. Parada, T. Peters, W. Sonke, and B. Speckmann. Optimal in-place compaction of sliding cubes. In *SWAT*, pages 31:1–31:14, 2024.

[13] C. Sung, J. Bern, J. Romanishin, and D. Rus. Reconfiguration planning for pivoting cube modular robots. In *ICRA*, pages 1933–1940. IEEE, 2015.

A Pseudocode of Compactify and Fix

Algorithm 1 Compactify(C)

Compute the slice graph of C.
while C is not quasi-compact (ignoring slice $z = 0$) **do**
  Let R be the cluster where the musketeers currently sit. DFS on the slice graph from R, preferring upward edges (in the z direction). Let S be the first extremal cluster (with possibly $S = R$).
  Move the musketeers to S.
  Lower non-quasi-compact modules in S as in Section 3.2.
  Let M be the set of stragglers $\mathbf{m}$ sorted by $\|\mathbf{m}\|_1$.
  for $\mathbf{m} \in M$ **do**
    Fix($\mathbf{m}$)
while $\exists \mathbf{m}$ where $\mathbf{m}+(0,0,-1)$ is empty and $z_{\mathbf{m}} > 0$ **do**
  Slide $\mathbf{m}$ down
if C is not compact **then**
  Apply the planar version of Compactify on slice $z = 0$

Algorithm 2 Fix($\mathbf{m}$)

while $\mathbf{m}$ can move monotonically in the direction $-\mathbf{m}$ **do**
  Move $\mathbf{m}$ closer to the origin
  if $\mathbf{m}$ moved to a lower z-coordinate or became quasi-compact **then**
    return
if $x_{\mathbf{m}} = 0$ or $y_{\mathbf{m}} = 0$ and $\mathbf{m}$ is not compact **then**
  Move $\mathbf{m}$ (through the plane $x = -1$ or $y = -1$) to a dominated empty cell p
else
  Let p be the closest dominated empty position to $\mathbf{m}$ where $x_p = x_{\mathbf{m}} - 1$ (resp., $y_p = y_{\mathbf{m}} - 1$), and $\mathbf{q}$ be the module at $p + (0,-1,0)$ (resp., $p + (-1,0,0)$)
  Move $\mathbf{q}$ to p and let q be the cell left empty originally occupied by $\mathbf{q}$
  for $i \in \{1, \ldots, (z_{\mathbf{m}} - z_q)\}$ **do**
    Move the module at $q+(0,0,i)$ to $q+(0,0,i-1)$ via slide
  Slide the module at $(x_p, y_p, z_{\mathbf{m}})$ to $\mathbf{m}+(-1,-1,0)$ and slide $\mathbf{m}$ to $(x_p, y_p, z_{\mathbf{m}})$
return