Reconfiguring Non-Convex Holes in Pivoting Modular Cube Robots

Daniel Feshbach and Cynthia Sung

***Abstract*—We present an algorithm for self-reconfiguration of admissible 3D configurations of pivoting modular cube robots with holes of arbitrary shape and number. Cube modules move across the surface of configurations by pivoting about shared edges, enabling configurations to reshape themselves. Previous work provides a reconfiguration algorithm for admissible 3D configurations containing no non-convex holes; we improve upon this by handling arbitrary admissible 3D configurations. The key insight specifies a point in the deconstruction of layers enclosing non-convex holes at which we can pause and move inner modules out of the hole. We prove this happens early enough to maintain connectivity, but late enough to open enough room in the enclosing layer for modules to escape the hole. Our algorithm gives reconfiguration plans with $\mathcal{O}\left(n^2\right)$ moves for n modules.**

***Index Terms*—Cellular and Modular Robots, Motion and Path Planning, Computational Geometry**

I. Introduction

SELF-RECONFIGURING robots or "programmable matter" smart materials, with the ability to change themselves into a wide variety of different shapes for different tasks or environments, would open substantial opportunities unavailable to state-of-the-art vehicle-like robots [1], [2]. Modular robotic systems attempt to tackle this challenge through a large number of modular units that attach and detach from one another and move around to form different shapes. A variety of motion models (e.g., sliding [3], [4], crystalline [5], [6], pivoting [7], [8], [9], [10], and hybrid multi-degree-of-freedom [11], [12]), connection strategies (e.g., active latching [13], passive mechanical latching [14], magnetic [15]), and module geometries (cubic [8], hexagonal [16], spherical [13]) exist for these systems.

For all of these systems, a major theoretical question is that of universal reconfigurability, or, equivalently, what types of shapes can be reconfigured into others. Motion planning for systems with a large number of modules is complex, particularly since in most modular systems, the modules are closely packed. Additional constraints on maintaining connectivity between modules to enable communication and coordination further complicates the problem.

Despite these challenges, theoretical guarantees of reconfigurability do exist for multiple types of systems. For example, for crystalline cube lattice systems, universal reconfiguration has been proven to be possible within $\mathcal{O}\left(n\right)$ steps [17], where n is the number of modules, and can actually be completed in $\mathcal{O}\left(\log n\right)$ time when multiple modules are allowed to move in parallel [18]. Similarly, sliding cubes in 2D [19] and 3D [7] have been shown to be universally reconfigurable in $\mathcal{O}\left(n^3\right)$ moves, and actively subtractable in 2D to a configuration with no holes in $\mathcal{O}\left(n^2\right)$ moves [20].

Manuscript received: February 24, 2021; Revised May 26, 2021; Accepted June 28, 2021.

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments.

The authors are with the General Robotics, Automation, Sensing and Perception (GRASP) Laboratory at the University of Pennsylvania, Philadelphia, PA 19104, USA. {feshbach, crsung}@seas.upenn.edu.

Digital Object Identifier (DOI): see top of this page.

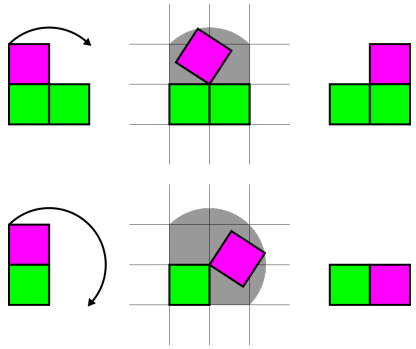

Fig. 1. Side view of cubes pivoting. Cubes pivot along shared edges until the face contacts another cube. Depending on cube arrangement this is a rotation of $\pi/2$ (first row) or π (second row). The gray region is the volume swept out by the moving module as it pivots: every cell intersecting the gray region must be empty.

A. Reconfiguration for Pivoting Cubes

In this paper, we are interested in the pivoting cube model: cubes move by pivoting across shared edges until contacting a face, as depicted in Fig. 1. A survey of existing hardware platforms (see above) indicates that rotation is the preferred method of motion in physical modular robots, considered relatively feasible to implement. It is therefore important to develop a theoretical characterization of pivoting cube systems, but this is currently much less explored than the siding and crystalline motion models, because pivoting has additional collision constraints which make universal reconfiguration guarantees elusive. In particular, when a module pivots, the volume it sweeps through includes more than just its initial and final positions. For a module to pivot without collision, all lattice cells (other than its start location) intersecting the volume the pivot sweeps through must also be empty. This adds a constraint not present for crystalline or sliding cube models (which require only the destination cell to be empty), so reconfiguration planning algorithms developed for these models are inapplicable to pivoting cubes. To further complicate the problem, because of these collision constraints,

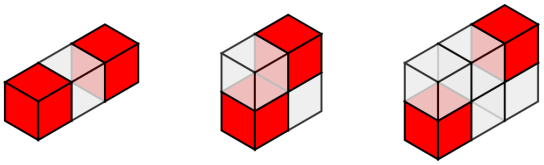

Fig. 2. Inadmissible sub-configurations defined in [21]: red cells are modules, white cells are empty. A configuration is admissible if it contains none of these three sub-configurations, referred to from left to right as rules 1, 2, and 3.

there exist infinitely many configurations that are "rigid"; that is, they cannot reconfigure at all [21].

Due to these challenges, it has been proven that for pivoting cube models, unlike sliding and crystalline models, deciding whether reconfiguration is possible between arbitrary configurations is PSPACE-hard [22]. Even when reconfiguration is possible, finding optimal plans is NP-complete for a version of the pivoting cube model allowing connected groups of modules to move together [23]. The existence of rigid configurations and challenge of deciding reconfigurability motivate a focus on sufficient but not necessary conditions for reconfigurability. Similar to how [24] handles hexagonal lattices, [21] defines a class of *admissible* configurations designed to ensure that a mobile cube exists on the outer boundary and that mobile cubes can freely traverse the surface, in 2D or 3D. Specifically, admissible configurations are defined as those that do not contain any instances of the subconfigurations in Fig. 2 (Fig. 3 and Fig. 4 depict problems that can arise from the inadmissible subconfigurations). By leveraging guarantees provable given admissibility, [21] proved reconfigurability for admissible 2D configurations, and for admissible 3D configurations without non-convex holes, providing reconfiguration plans in $\mathcal{O}\left(n^2\right)$ moves for n modules. However, the requirement that 3D configurations contain no non-convex holes requires global information to evaluate, which is undesirable. Follow-on work in [25] showed additionally that in 2D, if a configuration is not admissible, then as long as there are at least 5 free modules, reconfiguration is always possible.

As yet, the reconfigurability of general 3D configurations or of 2D configurations with fewer than 5 free modules remains unknown. As a result, many hardware platforms use a combination of external control signals [26] and heuristics [27]. Work in [28] takes a deep reinforcement learning approach to pivoting cubes reconfiguration, but lacks guarantees on the existence or cost of plans, and provides experimental results only for configurations too small (10 modules) to exhibit the kinds of complexity that make deciding reconfigurability challenging.

B. Our Contribution

In this paper, we consider the problem of reconfiguring any admissible n-module 3D configuration of pivoting cubes into any other. Our contribution is to prove that this is always possible, by giving an algorithm providing plans with $\mathcal{O}\left(n^2\right)$ moves. This result improves on [21] by handling holes of arbitrary shape, thereby expanding the class of known-reconfigurable configurations to one decidable using only local properties (admissibility conditions).

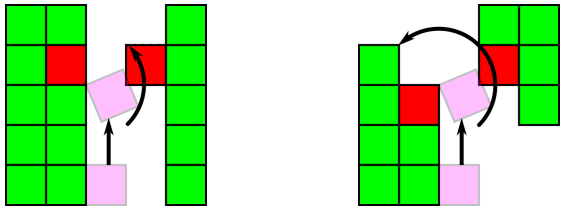

Fig. 3. Situations in which violations of rule 1 (left) and rule 3 (right) make it so a mobile module cannot freely traverse the configuration surface. The magenta module is the module traversing the surface, the red modules are the ones violating the rules.

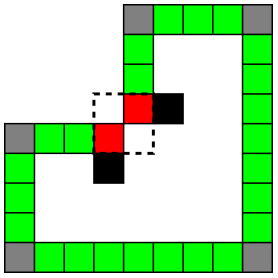

Fig. 4. A 2D configuration in which, due to a violation of rule 2, there does not exist a mobile module on the outer boundary [21]. The dashed line indicates the violation of rule 2. The gray modules on the outer corners have pivot moves available to them but would disconnect the configuration if removed. The black modules are mobile but not on the outer boundary.

The remainder of this paper is organized as follows. Section II presents the formal model of a pivoting cube modular system. Section III defines the reconfiguration problem and our main result: reconfigurability of arbitrary admissible 3D configurations. Section IV explains the corresponding reconfiguration algorithm for these structures. Section V provides the proofs of correctness and the complexity of the plans our algorithm provides. Section VI gives simulation evidence that our algorithm works. Section VII discusses remaining open questions and challenges for pivoting cube reconfiguration.

II. DEFINITIONS

Our robot model exists in a 3D cubic lattice, with connectivity defined along faces and cell coordinates represented in $\mathbb{Z}^3$. Axis choice is arbitrary: our model does not consider gravity or any environmental boundaries. Each cell may either be empty or occupied by a module. A *configuration* is a connected finite set of modules in the lattice.

A module can move itself around the surface of a configuration by pivoting around edges it shares with another module, as depicted in Fig. 1. Modules pivot until contacting the face of another module, after a rotation of either $\pi/2$ or π. In order to pivot, every cell (other than the start location) intersecting the volume swept out by the pivot must be empty: importantly, this includes more than just the destination cell. A module is *mobile* if there is a pivot it can perform to move to an adjacent cell without disconnecting the configuration.

A configuration is *admissible* if it does not contain any of the three *inadmissible sub-configurations* depicted in Fig. 2. The inadmissible sub-configurations are chosen to avoid situations which may inhibit modules from pivoting across a

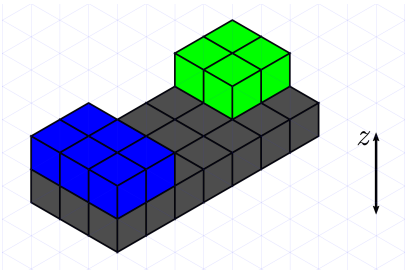

Fig. 5. The dark gray modules on the bottom layer are a single slice. In the top layer, the 6 blue modules on the left form a distinct slice from the 4 green modules on the right.

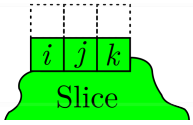

Fig. 6. Green cells are modules, white cells with dashed lines are empty. If module i is mobile in this layer, then i, j, k form a P_3 subconfiguration of the slice.

surface, and indeed, [21] proves that a mobile module can freely traverse the surface of an admissible configuration.

A *layer* refers to a plane with constant z coordinate. A *slice* of a configuration $\mathcal{C}$ is a maximally connected subset of a layer of $\mathcal{C}$. Fig. 5 exhibits the distinction between a layer and a slice. Since a slice is maximally connected within its layer, adjacent slices have distinct z-coordinates. Therefore we refer to extremity of slices based solely on the z coordinate of the slice, either locally (with respect to adjacent slices) or globally.

In contrast, referring to modules as extreme requires specifying a dimension, direction, or multiple directions in order of priority. For example, a (locally or globally) $+z$ extreme module is any module with (locally or globally) maximal z coordinate. Taking the $(+z, +y, +x)$ extreme of a configuration $\mathcal{C}$ means taking a subset $Z^+ \subseteq \mathcal{C}$ of globally $+z$ extreme modules, then the subset $Y^+ \subseteq Z^+$ of globally $+y$ extreme of Z^+, and finally the globally $+x$ extreme module of Y^+ (this is unique because Z^+ is constant in z so Y^+ is constant in z and in y).

Suppose the 2D boundary of a slice $\mathcal{S}$ contains three modules arranged consecutively in a line, all locally extreme in the same direction. If any of these modules have a pivot move available within this layer, we refer to the mobile module as i, the middle module as j, and the other end as j, and we call $\{i, j, k\}$ a P_3 subconfiguration of $\mathcal{S}$. Fig. 6 depicts this situation.

Let $\mathcal{S}$ be a locally extreme slice of $\mathcal{C}$, and $e_{\mathcal{C}}$ be some locally z-extreme module in $\mathcal{C}$. A *branch* of $\mathcal{S}$ (with respect to $\mathcal{C}$ and $e_{\mathcal{C}}$) is a maximally connected sub-configuration $\mathcal{B} \subseteq \mathcal{C} - \mathcal{S}$ which is connected to $e_{\mathcal{C}}$ only through $\mathcal{S}$. Note that $\mathcal{C} - \mathcal{S}$ is connected if and only if $\mathcal{S}$ has no branches. We call $\mathcal{S}$ the *anchor slice* of $\mathcal{B}$, and if $g \in \mathcal{S}$ and $b \in \mathcal{B}$ are adjacent modules, we call g an *anchor module* of $\mathcal{B}$ and b a *base* of $\mathcal{B}$.

A maximally connected set of empty cells entirely enclosed (in 3D, unless otherwise stated) by a configuration $\mathcal{C}$ is called a *hole* of $\mathcal{C}$. In the lattice, a *line* is a set of all cells fixed in two coordinates (e.g., the line fixing $x = x_0$, $y = y_0$ is $\{(x_0, y_0, z) : z \in \mathbb{Z}\}$). A hole $\mathcal{H}$ is *convex* if $\ell \cap \mathcal{H}$ is connected for every line ℓ. If $\mathcal{H}$ is *non-convex*, it must have a line along which it is disconnected, i.e., there must be at least one module in between line segments of $\mathcal{H}$ (segments of empty cells): such a module is called a *witness* to the non-convexity of $\mathcal{H}$.

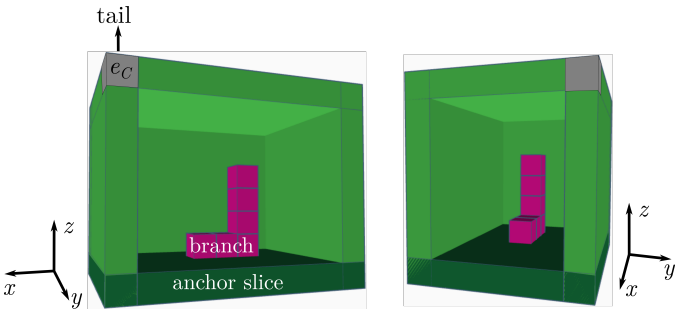

Fig. 7. An in-branch in a non-convex hole (two view angles with different sides transparent since the hole is closed in 3D). The dark green slice at the bottom is the anchor slice, the middle pink modules are the in-branch. Since $e_{\mathcal{C}}$ (the $(+z, +y, +x)$ extreme) is in the top slice, the bottom slice is the only slice which can be selected as next for deconstruction.

If a branch $\mathcal{B}$ of some slice in $\mathcal{C}$ is "inside of" (i.e., composed of modules witnessing the non-convexity of) a non-convex hole of $\mathcal{C}$, we call $\mathcal{B}$ an *in-branch* of $\mathcal{C}$. Fig. 7 depicts an in-branch in a non-convex hole.

III. PROBLEM AND RESULT STATEMENT

The problem that we aim to solve is as follows:

Problem 1. *Let $\mathcal{C}, \mathcal{C}'$ be admissible configurations of n pivoting modular cubes each. Is it possible to reconfigure $\mathcal{C}$ into $\mathcal{C}'$? If so, assuming global knowledge and centralized control, find a reconfiguration plan achieving this.*

A reconfiguration plan can be expressed as a sequence $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_k$ of configurations where each $\mathcal{C}_{i+1}$ differs from $\mathcal{C}_i$ by a valid pivot motion of one module. Similarly to [21], we exclude configurations containing any of three *inadmissible sub-configurations* – cube arrangements which create passages too narrow for a cube to pivot through – in order to guarantee that mobile modules can freely traverse the surface. The original work also excludes configurations containing non-convex holes. We now demonstrate that such an exclusion is not necessary, and that, in fact, all admissible 3D configurations are reconfigurable.

Our reconfiguration strategy is similar to most existing reconfiguration strategies in the literature, where modules are moved one by one to build up a linear "tail," eventually forming a line segment with all n modules in a total of $\mathcal{O}\left(n^2\right)$ pivots. To build the target configuration, it then reverses the plan which would turn that configuration into a line. Our main result is Theorem 1.

Theorem 1. *For any pair $\mathcal{C}, \mathcal{C}'$ of admissible configurations with n modules each, there exists a reconfiguration plan from $\mathcal{C}$ to $\mathcal{C}'$ that uses $\mathcal{O}\left(n^2\right)$ moves.*

Algorithm 1 shows our strategy for how to reconfigure any admissible configuration into a line using reversible pivots.

IV. ALGORITHM

A. Overall Strategy

To reconfigure an admissible configuration $\mathcal{C}$ into a line, we "deconstruct" $\mathcal{C}$ by "removing" modules one by one, pivoting them across the configuration surface to the end of the tail (which is built up from the $(+z, +y, +z)$ extreme module). The key is the order in which we choose modules to move to the tail. We select a slice to deconstruct next and remove its modules by proceeding in an order (mostly counterclockwise around the slice boundary) designed to maintain connectivity to the rest of the configuration and preserve admissibility. If the slice has a branch, we deconstruct until the branch has only one base slice, then remove the entire branch, then resume deconstructing the current slice.

Algorithm 2 defines the order within a slice to remove modules: this order follows [21]. Algorithm 3 defines the order of module removal within the whole 3D configuration, including calls to Algorithm 2 and recursive calls on branches. Algorithm 1 takes the modules in the order given by Algorithm 3 and sends them to the end of the tail.

B. Order of Removal Within a Slice

In order to maintain connectivity of the configuration, when we select a slice we mark some module as the *root of deconstruction*, based on the following condition, and make sure to remove it last.

Condition 1 (Root Module Selection)**.** *A module $r \in \mathcal{S}$ is eligible to be the root of deconstruction (with respect to the $\mathcal{C}, e_{\mathcal{C}}$ at this level of recursion) if there exists a path in the module-connectivity graph of $\mathcal{C}$ from r to $e_{\mathcal{C}}$ which uses no modules in $\mathcal{S}$ other than r.*

Since branches are connected to the rest of $\mathcal{C}$ only through their anchor slice, this condition ensures that either r connects $\mathcal{S}$ to another slice $\mathcal{S}'$ which is not within a branch of $\mathcal{S}$, or $r = e_{\mathcal{C}}$.

Given a slice $\mathcal{S}$ and its assigned root r, Algorithm 2 builds up a list of the modules in $\mathcal{S}$ in the order they will be removed from the slice and pivoted to the end of the tail of $\mathcal{C}$. This order is identical to that used in [21]. Working counterclockwise from a corner module e (arbitrary, the $(+y, +x)$ extreme), if we encounter a mobile module i which can be removed while preserving admissibility of the slice, we remove that. If we encounter a P_3 subconfiguration, we remove its modules in order i, j, k (the point of P_3 subconfigurations is that removing i might temporarily break an admissibility condition, but then removing j and k is guaranteed to restore it [21]). We avoid selecting e or r for removal unless they are the only mobile modules, in which case we remove e and update it to be the new extreme; we only select r when it is the last module remaining.

C. Selecting Slices

The next slice to be deconstructed is selected as any which meets the following conditions.

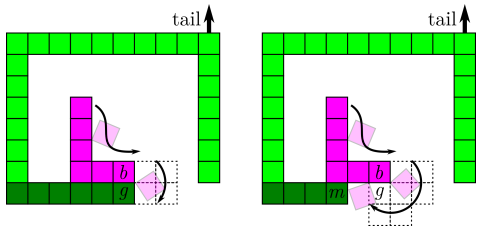

Fig. 8. A moment in the deconstruction of the configuration depicted in Fig. 7, side view. When deconstructing the bottom (anchor) slice, we will pause just before the next module to be removed is some anchor module m, and remove the entire L shaped pink branch. There are two cases depending on whether m is the same module as g, the first anchor module here selected for removal (the base module b corresponding to g is also labeled because it is referenced in the proof). If $m = g$ (left figure) we can move the branch modules next to g one by one, then out to the tail. If $m \neq g$ (right figure), then g has already been removed by the time we pause to deconstruct the branch, so we can reach the cell where g was. Since g and m are anchor modules of the same slice of the in-branch, we can traverse the bottom of this slice to reach the spot next to m, and from there we know we can reach the tail.

Condition 2 (Slice Selection)**.** *A slice $\mathcal{S}$ of configuration $\mathcal{C}$ with extreme module $e_{\mathcal{C}}$ is eligible to be deconstructed next if:*

(a) $\mathcal{S}$ is locally extreme in z
(b) $\mathcal{S}$ is on the outer boundary of $\mathcal{C}$
(c) If $\mathcal{C}$ has multiple slices, then $\mathcal{S}$ does not include $e_{\mathcal{C}}$

For comparison, [21] required (a) and (c) and also that $\mathcal{C} - \mathcal{S}$ is connected (i.e., that $\mathcal{S}$ has no branches). Absent non-convex holes (a) implies (b), but with non-convex holes, we need condition (b) to avoid selecting slices whose modules are trapped within holes and cannot reach the tail.

The presence of non-convex holes sometimes means every slice satisfying (a), (b), and (c) has at least one branch. For example, in the configuration in Fig. 7, the top and bottom slices are the only locally extreme slices on the outer boundary, but the bottom slice contains the extreme module, so only the top slice (which has the pink modules as an in-branch) can be selected. Never selecting slices with branches allows [21] to simply remove entire slices all at once before proceeding to the next one, but if we did that we would disconnect branches.

D. Handling Branches

Say we are deconstructing a slice $\mathcal{S}$ which has branches. When the next module $m \in \mathcal{S}$ to be removed from $\mathcal{S}$ is an anchor module of branch $\mathcal{B}$, and specifically is connected to the only slice $\mathcal{S}_B$ of $\mathcal{B}$ still connecting $\mathcal{B}$ to what remains of $\mathcal{S}$, we pause deconstruction of $\mathcal{S}$ to avoid risking disconnecting $\mathcal{B}$. Then we deconstruct $\mathcal{B}$ entirely, in order given by a recursive call to Algorithm 3, and then resume deconstruction of $\mathcal{S}$ at m where we left off.

The specified extreme module $e_{\mathcal{C}}$ tells Algorithm 3 which slice to deconstruct last, and which module to remove last from that slice. For the global configuration (the outermost call), $e_{\mathcal{C}}$ is the top corner module, which we build the tail upwards from. For a recursive call on branch $\mathcal{B}$, $e_{\mathcal{C}}$ is a base connecting $\mathcal{B}$ to the rest of the configuration.

Algorithm 1: RECONFIGURETOLINE($\mathcal{C}_0$)

Input: admissible configuration $\mathcal{C}_0$
Output: sequence of configurations transforming $\mathcal{C}_0$ into a line using reversible pivots
$e \leftarrow (+z, +y, +x)$ extreme module of $\mathcal{C}_0$
$i \leftarrow 0$
for m *in* REMOVALORDER3D($\mathcal{C}_0, e$) **do**
  $P \leftarrow$ reversible path from m to $(e_x, e_y, e_z + 1)$ given by breadth-first search
  for *each step in* P **do**
    $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_i$ with m pivoted one step along P
    $i \leftarrow i + 1$
  end
  $e \leftarrow (e_x, e_y, e_z + 1)$ `// new end of tail`
end
return $\mathcal{C}_0, \ldots, \mathcal{C}_i$

Algorithm 2: REMOVALORDER2D($\mathcal{S}, r$)

Input: An admissible 2D configuration $\mathcal{S}$, a module r in $\mathcal{S}$
Output: A list ordering the modules of $\mathcal{S}$, with r last, in an order they can remove themselves without disconnecting $\mathcal{S}$
$e \leftarrow (+y, +x)$ extreme module of $\mathcal{S}$
$L \leftarrow$ empty list
while $\mathcal{S}$ *is not empty* **do**
  $M \leftarrow$ mobile modules on the boundary of $\mathcal{S}$, listed in counterclockwise order starting counter-clockwise of e, excluding e and r
  if *M is empty* **then**
    Append e to L
    $\mathcal{S} \leftarrow \mathcal{S} - \{e\}$
    $e \leftarrow (+y, +x)$ extreme module of $\mathcal{S}$
  else
    $i \leftarrow$ first module of M such that $\mathcal{S} - \{i\}$ is admissible OR i in an instance of P_3
    Append i to L
    if $\mathcal{S} - \{i\}$ *is not admissible* **then**
      Append i to L
      $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
    else
      Find j, k such that (i, j, k) is an instance of P_3
      Append i, j, k to L in order;
      $\mathcal{S} \leftarrow \mathcal{S} - \{i, j, k\}$
    end
  end
end
return L

Algorithm 3: REMOVALORDER3D($\mathcal{C}, e_\mathcal{C}$)

Input: An admissible configuration $\mathcal{C}$, a locally z-extreme module $e_\mathcal{C} \in \mathcal{C}$.
Output: A list of the modules of $\mathcal{C}$, with $e_\mathcal{C}$ last, in an order they can move themselves to the end of the tail without disconnecting $\mathcal{C}$.
$R \leftarrow$ empty list `// removal order list`
while $\mathcal{C}$ *is not empty* **do**
  $\mathcal{S} \leftarrow$ slice of $\mathcal{C}$ satisfying Condition 2
  $L_B \leftarrow$ list of branches of $\mathcal{S}$
  if $e_\mathcal{C} \in \mathcal{S}$ **then**
    $r \leftarrow e_\mathcal{C}$
  else
    $r \leftarrow$ some module of $\mathcal{S}$ satisfying Condition 1
  end
  for m *in* REMOVALORDER2D($\mathcal{S}, r$) **do**
    if m *has a neighbor* b *in some branch* $\mathcal{B} \in L_B$, *AND the slice containing* b *is the only slice of* $\mathcal{B}$ *adjacent to current* $\mathcal{S}$ **then**
      `// remove` $\mathcal{B}$ `before` m
      $R \leftarrow$ concatenate REMOVALORDER3D($\mathcal{B}, b$)) to the end of R
      remove $\mathcal{B}$ from $\mathcal{C}$ and L_B
    end
    append m to R
    remove m from $\mathcal{S}$ and $\mathcal{C}$
  end
end

V. PROOFS

Our algorithm differs from [21] in modifications to the conditions to select the slice and its root of deconstruction, and by pausing the deconstruction of certain slices while removing their branches. As such, we need to prove that slices and roots can still always be selected, and that we pause deconstruction of slices early enough to maintain connectivity but late enough that in-branch modules can now escape the hole and reach the tail.

A. Connectivity, Slice Selection, and Root Selection

First, it will be useful to prove that we do not have to worry about the configuration becoming disconnected. We have yet to prove that our algorithm does not get "stuck" (by being unable to select a slice or root, or by being unable to find a path from a selected module to the tail), but we can nonetheless prove that as long as it has been running so far, the configuration must still be connected.

Lemma 1 (Connectivity)**.** *As long as Algorithm 1 runs, the modules that it is reconfiguring remain connected (except that while a module pivots it is only edge-connected).*

Proof. We will proceed by induction on the number of slices removed and on the depth of recursion. The base case for number of slices removed is that the configuration fed to that level of recursion starts out connected, so we assume whenever beginning deconstruction of slice $\mathcal{S}$ that the current configuration is connected. The base case for recursive depth is the innermost recursive call, which never selects a slice while it still has branches (because that would trigger a deeper recursive call), so we remove entire slices at a time without disconnecting the configuration. Then we assume as

an inductive hypothesis that connectivity of branches is always maintained within recursive calls.

Given a slice $\mathcal{S}$ next to be deconstructed, we know from [21] that it remains internally connected throughout its deconstruction, and that it remains connected to the external configuration (i.e., excluding any branches it may have). So for the inductive step, we need to show that $\mathcal{S}$ remains connected to its branches. The recursive call to any branch must happen before the removal of its last anchor module (because the last anchor module removed would be connected to the last branch slice connected to what remains of $\mathcal{S}$), so at the beginning of the recursive call everything is still connected. Then during the recursive call the branch remains internally connected, and since the extreme module (which will be the last module to be removed) used in the recursive call is a base, it remains connected to $\mathcal{S}$ throughout the call. Finally, once we have removed all branches of $\mathcal{S}$, removing the rest of $\mathcal{S}$ cannot disconnect the configuration (this is the point of the definition of branch). □

Now we can start seeing why the algorithm will not get stuck unable to select something.

Lemma 2 (Slice Selection)**.** *Any admissible configuration $\mathcal{C}$ contains a slice $\mathcal{S}$ that satisfies Condition 2.*

Proof. Any globally extreme slice is locally extreme and is on the outer boundary. There must be at least two globally extreme slices, only one of which contains the extreme module, so the other satisfies (a), (b), and (c). □

Lemma 3 (Root Module Selection)**.** *If a slice $\mathcal{S}$ is selected for deconstruction, then it must contain a module r satisfying Condition 1.*

Proof. If $e_{\mathcal{C}} \in \mathcal{S}$ (i.e. if $\mathcal{C}$ has only one slice), $r = e_{\mathcal{C}}$ has a trivial path to itself not using any other modules in the slice. Otherwise, since the configuration remains connected (Lemma 1), there must be a shortest path in the slice graph from $\mathcal{S}$ to the $e_{\mathcal{C}}$. Let $\mathcal{S}'$ be the next slice after $\mathcal{S}$ on that path. Since $\mathcal{S}'$ and $\mathcal{S}$ are neighbors in the slice graph, there exists some $r \in \mathcal{S}$ adjacent to a module in $\mathcal{S}'$. Since shortest paths contain no cycles, the remainder of the path in the slice graph defines a path in the module connectivity graph connecting r to $e_{\mathcal{C}}$ without passing through $\mathcal{S}$ again. □

B. Removal of Branches

It remains to show that whenever Algorithm 3 selects a module m, there always exists a path of reversible pivots from m to the tail. The only case not covered by surface traversability guarantees from [21] is modules selected within recursive calls to branches. We will show that the recursive call happens late enough in the deconstruction of anchor slices that a spot has opened up at which modules can pivot from a position on the branch to one on the anchor slice.

While deconstructing a slice $\mathcal{S}$ with a branch $\mathcal{B}$, let $m \in \mathcal{S}$ be a module triggering a recursive call on $\mathcal{B}$, i.e., an anchor module of some branch $\mathcal{B}$ whose corresponding base module is part of the only slice $\mathcal{S}_b$ of $\mathcal{B}$ adjacent to $\mathcal{S}$. Let $g \in \mathcal{S}$ be the *first* anchor module of $\mathcal{S}_b$ removed from $\mathcal{S}$ (it is possible that $g \neq m$, since g may have been removed at a time when $\mathcal{B}$ still had multiple base slices). Write coordinates relative to $g = (0, 0, 0)$. Without loss of generality say $\mathcal{S}$ is a minimum, so the corresponding base module is $b = (0, 0, 1)$.

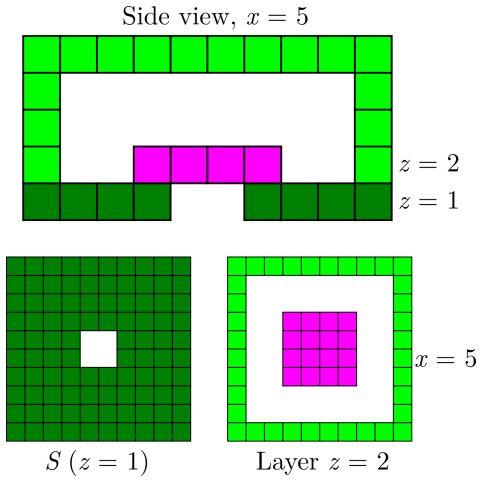

Fig. 9. It is possible that a slice $\mathcal{S}$ with a 2D hole could have an in-branch, if a slice of the in-branch covers just above/below the 2D hole to close up a 3D hole. In this situation, admissibility rule 2 guarantees that each module of $\mathcal{S}$ adjacent to the 2D hole also have a module of the in-branch just above/below it.

A *step* in the deconstruction of a slice refers either to the removal of one mobile module whose removal preserves admissibility of the slice, or to the removal of a P_3 subconfiguration: steps preserve slice admissibility [21]. Let $\mathcal{S}'$ be the portion of $\mathcal{S}$ remaining just before the step in which g would be removed: then $\mathcal{S}'$ is admissible with g on its 2D outer boundary.

Lemma 4. *There exists an x or y direction in which both g and b are locally extreme in $\mathcal{S}'$ and $\mathcal{B}$ respectively.*

Proof. Since g will be removed in the next step of deconstructing $\mathcal{S}$, it is either mobile or part of a P_3 sub-configuration, so it is locally extreme within $\mathcal{S}'$. Without loss of generality say $+x$ is an outwards direction from g, i.e., $(1, 0, 0)$ is empty and is not in a 2D hole of $\mathcal{S}'$. We will show that b is also locally $+x$ extreme.

Suppose for contradiction that b is not $+x$ extreme, i.e., $(1, 0, 1)$ is a module. This is a module adjacent to b within its layer, i.e., part of the same slice containing b. Then $(1, 0, 0) \notin \mathcal{S}$, because we know it is not currently in $\mathcal{S}'$, so if it had once been a module it would also have been an anchor module of $\mathcal{S}_b$, contradicting the assumption that g was the earliest such anchor module removed. So $(1, 0, 0)$ must have been in a 2D hole $\mathcal{H}_{\mathcal{S}}$ of $\mathcal{S}$.

Note that $\mathcal{H}_{\mathcal{S}}$ must have been "plugged" by modules directly above each position (or else $\mathcal{B}$ would not be an in-branch because $\mathcal{S}$ would not be enclosing a 3D hole), as in Fig. 9. Furthermore, admissibility rule (2) guarantees that every position adjacent to the boundary of a 2D hole in $\mathcal{S}$ must also have a module in $\mathcal{B}$ above it.

Starting with $(1, 0, 0)$, trace the 2D surrounding set of $\mathcal{S}'$ in either direction. If you encounter an empty cell $(p, q, 0)$ under a module $(p, q, 1)$ on the outer boundary of $\mathcal{S}_b$, then (since the

3D hole was originally closed and by admissibility rule 2) we must have $(p, q, 0) \in \mathcal{S}$, contradicting that g is the first anchor module of $\mathcal{S}_b$ selected. Otherwise, tracing the surrounding set loops back around to $(1, 0, 0)$ without ever passing over the outer boundary of $\mathcal{S}_b$, so $\mathcal{H}_\mathcal{S}$ is still a closed 2D hole in $\mathcal{S}'$, contradicting the choice of $+x$ as an *outwards* direction of extremity from g. □

Now that we know this direction exists from g, we can refer to it as $+x$ without loss of generality, and say $(1, 0, 0)$ and $(1, 0, 1)$ are empty. The next step is to show that there are enough empty cells in this direction for a module from $\mathcal{B}$ to pivot into the $z = 0$ layer here.

Lemma 5. *When $\mathcal{S}'$ is what remains of $\mathcal{S}$ (i.e. when g will be removed in the next step of deconstruction of $\mathcal{S}$), a mobile module on $\mathcal{B}$ can move to $(1, 0, 1)$ and then pivot to $(1, 0, 0)$.*

Proof. The global configuration starts out as admissible, and [21] proves both that the removal of entire slices preserves admissibility and that admissibility within slices with respect to themselves is preserved between deconstruction steps. Therefore the only admissibility violations that could currently exist are between $\mathcal{S}'$ and $\mathcal{B}$; we still know $\mathcal{B}$ is admissible with respect to itself and its surroundings below $\mathcal{S}'$, and that $\mathcal{S}'$ is admissible with respect to itself and its surroundings within its own layer. Now, since $g = (0, 0, 0)$ and $b = (0, 0, 1)$ are modules and $(1, 0, 0)$ and $(1, 0, 1)$ are empty, admissibility rule 1 guarantees that $(2, 0, 0)$ and $(2, 0, 1)$ must also be empty.

It was shown in [21] that mobile modules can freely traverse the surface of admissible configurations, so admissibility of $\mathcal{B}$ means that modules selected from it can move to $(1, 0, 1)$. Then emptiness of $(2, 0, 0)$ and $(2, 0, 1)$ allows them to pivot to $(1, 0, 0)$. □

The key that makes this useful is that $\mathcal{S}'$ characterizes the earliest possible point when the recursive call to $\mathcal{B}$ could begin: if the call happens later, even more will have been removed from $\mathcal{S}$, making it that much easier for modules to escape the hole.

Lemma 6. *Any module selected by Algorithm 3 has a path to reach the tail.*

Proof. If a module was selected from a slice at the outermost call to Algorithm 3, [21] proves it can reach the current $(+y, +z)$ extreme of its slice and then move to the tail from there. Otherwise, this module is selected within a recursive call to a branch: continue previous notation for this case. If $m = g$ (i.e. if the anchor module triggering the recursive call is the first anchor module of the corresponding base slice selected by Algorithm 2), Lemma 5 guarantees that all modules until the deconstruction of $\mathcal{S}_b$ can go to $(1, 0, 1)$, pivot to $(1, 0, 0)$, and then can reach the tail. See the left part of Fig. 8 for an illustration.

If $m \neq g$, then the recursive call is triggered after g has already been removed. So until $\mathcal{S}_b$ starts to be deconstructed, modules can go to $(1, 0, -1)$ and make a π pivot to g (the additional cells passed through at the $z = 1$ layer are also empty because $\mathcal{S}$ was selected as a minimum), then traverse below $\mathcal{S}_b$ to a mobile position on the boundary of what remains of $\mathcal{S}$, from which it is possible to reach the tail. See the right part of Fig. 8 for an illustration.

During the deconstruction of $\mathcal{S}_b$, there is at least as much room as before for modules to move to the $z = 0$ layer. Once they do so they can traverse the bottom of $\mathcal{S}_b$ (which remains connected) to reach some cell adjacent to what remains of $\mathcal{S}$, from which point they can reach the tail. □

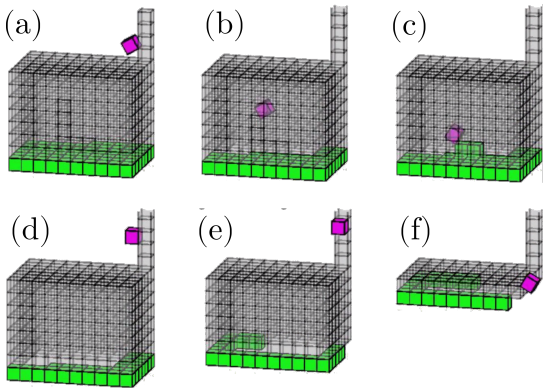

Fig. 10. Simulation for the configuration in Fig. 7 with an L-shaped in-branch (our attached video animates this). The current slice, and paused anchor slices, are in light green. The module moving to the tail is in magenta. All other modules are transparent gray. We select the bottom slice and move its modules one-by-one to the tail (as in (a)) until the next module is below the branch. Then we select and remove slices top-down from the branch (seen in (b) and (c)) until the branch has been removed, after which we resume removing modules from the bottom slice (as in (d)). Finally, we select slices upwards (as in (e) and (f)) to move the rest of the configuration to the tail.

Finally, we can prove Theorem 1.

Proof. Lemma 1 guarantees that whenever we remove the next module selected by Algorithm 3, we will not be disconnecting the configuration. Lemmas 2 and 3, along with guarantees from [21] about the order of deconstruction within a slice (which we do not alter), guarantee that Algorithm 3 runs to completion. Then Lemma 6 shows that every module it yields has a path to the tail, i.e., that Algorithm 1 runs to completion. This reconfigures any admissible configuration into a line; since plans are reversible, the line can then reconfigure into any other admissible configuration with the same number of modules.

As in [21], plans take $\mathcal{O}(n^2)$ steps to reconfigure a configuration of n modules. The upper bound is from the fact that n modules take non-cyclical paths (at most n pivots) to the tail. A lower bound is given by a configuration of n modules stacked vertically, requiring $\Omega(n^2)$ pivots to form a line. □

VI. SIMULATION

We implement our reconfiguration algorithm in Python and animate the resulting plans using MATLAB. We test our planner on several configurations, including situations involving recursion to in-branches as well as recursion to branches that are on the outside in the first place. Fig. 10 shows frames of the plan given for the configuration from Fig. 7: the video attached in publication shows an animation of this plan. The animation video is additionally available at `https://youtu.be/U8UFYt6CTQo`.

VII. CONCLUSION

We prove reconfigurability for arbitrary admissible 3D configurations, extending upon previous results showing reconfiguration only for configurations with non-convex holes. Since the existence of non-convex holes is a global property but admissibility conditions are local, we have taken a significant step towards distributing planning across the module network, and substantially reduced the amount of global reasoning that a higher-level system (e.g. a task planner selecting desired configurations) would have to do. Guaranteeing the existence of reconfiguration plans is also a useful prerequisite to building more sophisticated planning systems.

One immediate area for future work is to further expand the configurations on which we guarantee reconfigurability by relaxing admissibility conditions. This could include characterizing the situations in which each admissibility rule is actually necessary, or using a fixed number of mobile helper modules as [25] does in 2D.

Another important next step is to allow modules to move in parallel. We expect that with proper communication to avoid collision, one could reconfigure in $\mathcal{O}(n)$ time steps, with possible exception for limited bottleneck situations. Much of our approach could be parallelized (e.g. by selecting multiple eligible slices to deconstruct at once). Further work could consider how to distribute the computation of the plans across the module network.

Finally, now that we know reconfigurability is possible for all admissible configurations, future work should attempt to find more direct plans: always passing through an intermediate line state is highly suboptimal in many cases. While we suspect computing optimal plans to be NP-hard in general even for admissible configurations, it may be possible to formally characterize similarity between configurations in a way which makes optimality tractable in restricted cases, allows heuristic approaches with formal bounds, or provides helpful structure for learning-based systems.

REFERENCES

[1] J. Seo, J. Paik, and M. Yim, "Modular reconfigurable robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 63–88, 2019. 1

[2] M. Yim, W.-m. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems [Grand challenges of robotics]," *Robotics Automation Magazine, IEEE*, vol. 14, pp. 43 – 52, 04 2007. 1

[3] M. Koseki, K. Minami, and N. Inou, "Cellular robots forming a mechanical structure (evaluation of structural formation and hardware design of "CHOBIE II"," in *Distributed Autonomous Robotic Systems 6*, 2004, pp. 139–148. 1

[4] B. K. An, "Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 3149–3155. 1

[5] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Structure decision method for self organising robots based on cell structures-cebot," in *IEEE International Conference on Robotics and Automation*, 1989, pp. 695–696. 1

[6] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, no. 1, pp. 107–124, 2001. 1

[7] Z. Abel and S. D. Kominers, "Universal reconfiguration of (hyper-) cubic robots," *ArXiv preprint*, p. arXiv:0802.3414, 2008. 1

[8] J. Romanishin, K. Gilpin, S. Claici, and D. Rus, "3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions," *IEEE International Conference on Robotics and Automation*, vol. 2015, pp. 1925–1932, 06 2015. 1

[9] A. B. Zia, S. O. Ejaz, S. Abbas, U. Ikram, and S. ur Rehman, "Mucubes: Modular, cube shaped, and self-reconfigurable robots," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, Nov 2017, pp. 1–4. 1

[10] S. Hauser, M. Mutlu, and A. J. Ijspeert, "Kubits: Solid-state self-reconfiguration with programmable magnets," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6443–6450, Oct 2020. 1

[11] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 431–441, 2002. 1

[12] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots-design of the smores system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4464–4469. 1

[13] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A. J. Ijspeert, "Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1016–1033, 2014. 1

[14] N. Eckenstein and M. Yim, "Design, principles, and testing of a latching modular robot connector," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2846–2851. 1

[15] N. Ayanian, P. J. White, A. Hálász, M. Yim, and V. Kumar, "Stochastic control for self-assembly of XBots," in *ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2008, pp. 1169–1176. 1

[16] S. Murata, H. Kurokawa, and S. Kokaji, "Self-assembling machine," in *IEEE International Conference on Robotics and Automation*, 1994, pp. 441–448. 1

[17] 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*, vol. 42, no. 6-7, pp. 652–663, 2009. 1

[18] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer, "Reconfiguration of cube-style modular robots using o (logn) parallel moves," in *International Symposium on Algorithms and Computation*, 2008, pp. 342–353. 1

[19] A. Dumitrescu and J. Pach, "Pushing squares around," in *Annual Symposium on Computational Geometry*, 2004, pp. 116–123. 1

[20] M. D. Hall, A. Özdemir, and R. Groß, "Self-reconfiguration via active subtraction with modular robots," in *Proceedings of Robotics: Science and Systems*, 2020. 1

[21] C. Sung, J. Bern, J. Romanishin, and D. Rus, "Reconfiguration planning for pivoting cube modular robots," in *2015 IEEE International Conference on Robotics and Automation*, 2015, pp. 1933–1940. 2, 3, 4, 5, 6, 7

[22] 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," *ArXiv preprint*, p. arXiv:2012.07556, 2020. 2

[23] Z. Ye, M. Yu, and Y.-J. Liu, "NP-completeness of optimal planning problem for modular robots," *Autonomous Robots*, vol. 43, no. 8, pp. 2261–2270, 2019. 2

[24] A. Nguyen, L. J. Guibas, and M. Yim, "Controlled module density helps reconfiguration planning," in *International Workshop on Algorithmic Foundations of Robotics*, 2000, pp. 23–36. 2

[25] H. A. Akitaya, E. M. Arkin, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, M. Korman, B. Palop, I. Parada, A. van Renssen, *et al.*, "Universal reconfiguration of facet-connected modular robots by pivots: the O(1) musketeers," *Algorithmica*, 2021. 2, 8

[26] J. W. Romanishin, J. Mamish, and D. Rus, "Decentralized control for 3D M-Blocks for path following, line formation, and light gradient aggregation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 4862–4868. 2

[27] P. Thalamy, B. Piranda, and J. Bourgeois, "A survey of autonomous self-reconfiguration methods for robot-based programmable matter," *Robotics and Autonomous Systems*, vol. 120, p. 103242, 2019. 2

[28] Q. Song, D. Ye, Z. Sun, and B. Wang, "Autonomous reconfiguration of homogeneous pivoting cube modular satellite by deep reinforcement learning," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 2020. 2