Linear Reconfiguration of Cube-Style Modular Robots☆

Greg Aloupis[a], Sébastien Collette[a,1], Mirela Damian[b], Erik D. Demaine[c], Robin Flatland[d], Stefan Langerman[a,2], Joseph O'Rourke[e], Suneeta Ramaswami[f,3], Vera Sacristán[g,4], Stefanie Wuhrer[h]

[a] *Université Libre de Bruxelles, Belgique*
[b] *Villanova University, Villanova, USA*
[c] *Massachusetts Institute of Technology, Cambridge, USA*
[d] *Siena College, Loudonville, USA*
[e] *Smith College, Northampton, USA*
[f] *Rutgers University, Camden, USA*
[g] *Universitat Politècnica de Catalunya, Barcelona, Spain*
[h] *Carleton University, Ottawa, Canada*

Abstract

In this paper we propose a novel algorithm that, given a source robot S and a target robot T, reconfigures S into T. Both S and T are robots composed of n atoms arranged in $2 \times 2 \times 2$ meta-modules. The reconfiguration involves a total of $O(n)$ atomic operations (expand, contract, attach, detach) and is performed in $O(n)$ parallel steps. This improves on previous reconfiguration algorithms [1, 2, 3], which require $O(n^2)$ parallel steps. Our algorithm is in-place; that is, the reconfiguration takes place within the union of the bounding boxes of the source and target robots. We show that the algorithm can also be implemented in a synchronous, distributed fashion.

Key words: self-reconfiguring modular robots, cubical units, in-place reconfiguration, lattice reconfiguration, synchronous distributed module communication

---

☆A short version appeared at ISAAC 2007 [4].

Email addresses: greg@scs.carleton.ca (Greg Aloupis), sebastien.collette@ulb.ac.be (Sébastien Collette), mirela.damian@villanova.edu (Mirela Damian), edemaine@mit.edu (Erik D. Demaine), flatland@siena.edu (Robin Flatland), stefan.langerman@ulb.ac.be (Stefan Langerman), orourke@cs.smith.edu (Joseph O'Rourke), rsuneeta@camden.rutgers.edu (Suneeta Ramaswami), vera.sacristan@upc.edu (Vera Sacristán), swuhrer@scs.carleton.ca (Stefanie Wuhrer)

[1] Chargé de Recherches du FNRS.
[2] Chercheur Qualifié du FNRS.
[3] Partially supported by NSF grant CCR-0204293.
[4] Partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

1. Introduction

A self-reconfiguring modular robot consists of a large number of independent units that can rearrange themselves into a structure best suited for a given environment or task. For example, it may reconfigure itself into a thin, linear shape to facilitate passage through a narrow tunnel, transform into an emergency structure such as a bridge, or surround and manipulate objects in outer space. Since modular robots comprise groups of identical units, they can also repair themselves by replacing damaged units with functional ones. Such robots are especially well-suited for working in unknown and remote environments.

Various types of units for modular robots have been designed and prototyped in the robotics community. These units differ in shape and the operations they can perform. In this paper, we consider homogeneous self-reconfiguring modular robots composed of cubical units (*atoms*) arranged in a lattice configuration. Each *atom* is equipped with an expansion/contraction mechanism that allows it to extend its faces out and retract them back. Each atom face has an attaching/detaching mechanism that allows it to attach to (or detach from) the face of an adjacent atom. Prototypes of cubical atoms include crystalline atoms [5] and telecube atoms [6]. The collection of atoms composing a robot is *connected* in the sense that its dual graph (vertices correspond to atoms, edges correspond to attached atoms) is connected. When groups of atoms perform the four basic *atom operations* (expand, contract, attach, detach) in a coordinated way, the atoms move relative to one another, resulting in a reconfiguration of the robot. To ensure connectedness of the reconfiguration space, the atoms are grouped into *meta-modules* [1, 2], which are connected sets of ℓ^3 atoms arranged in an $\ell \times \ell \times \ell$ grid.

The complexity of a reconfiguration algorithm can be measured by the number of *parallel steps* performed, as well as the total number of atom operations. In a parallel step, many atoms may perform moves simultaneously. Reducing the number of parallel steps has a significant impact on the reconfiguration time, because the mechanical actions (expand, contract, attach, detach) performed by the atoms are typically the slowest part of the system. Furthermore, since atoms may have limited battery power, it is useful to reduce the total number of mechanical operations (i.e., the atom operations) performed.

Our main contribution in this paper is a novel algorithm that, given a source robot S and a target robot T, each composed of n atoms arranged in $2 \times 2 \times 2$ meta-modules,[5] reconfigures S into T in $O(n)$ parallel steps and a total of $O(n)$ atomic operations. Our algorithm improves significantly the previously best-known reconfiguration algorithms for cube-style modular robots [1, 2, 3], which take $O(n^2)$ parallel steps as well as $O(n^2)$ atomic operations. In addition, our algorithm reconfigures S into T in-place, in the sense that the reconfiguration takes place within the union of the bounding boxes of S and T, while keeping

[5] Throughout the paper, n refers to the number of robot atoms and m refers to the number of robot meta-modules, where $n = 8m$.

the robot connected at all times during the reconfiguration. An in-place reconfiguration is useful when there are restrictions on the amount of space that a robot may occupy during the reconfiguration process. Note that in this work we have not taken into consideration any issues regarding the robot's mass or inertia. However, the "in-place" nature of our algorithms mitigates some of the issues arising from such constraints.

2. Preliminaries

2.1. Robots as Lattices of Meta-Modules

There exist atom configurations which cannot be reconfigured, e.g., a single row of atoms. Connectedness of the reconfiguration space can be guaranteed, however, for robots composed of meta-modules. It is desirable that meta-modules are composed of as few atoms as possible. In our reconfiguration algorithms, meta-modules are of minimum size, i.e. a $2 \times 2 \times 2$ grid of atoms [2, 7].

We define two basic meta-module moves (hardware independent) used by our reconfiguration algorithms, similar to the ones described in [2].

Slide(***dirSlide***). Slides a meta-module one step in the direction *dirSlide* with respect to some substrate meta-modules. This move is illustrated in Figure 1, where each box represents a meta-module. The preconditions for applying this move are: (i) the sliding meta-module (A in Figure 1a) is adjacent to a meta-module in a direction orthogonal to *dirSlide* (B in Figure 1a), which in turn is adjacent to a meta-module in direction *dirSlide* (C in Figure 1a) and (ii) the target position for the sliding meta-module is free. This move allows the sliding meta-module to "carry" other attached

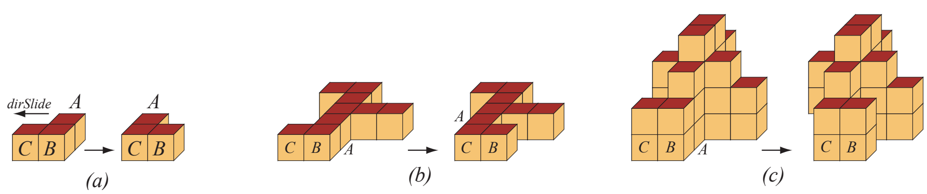

Figure 1: Examples of Slide(x^-): (a) Meta-module A slides alone, (b,c) A carries adjacent meta-modules.

meta-modules (as in Figs. 1b-c), as long as the target positions for the carried meta-modules are unoccupied.

k-Tunnel(***sPos*, *ePos***). Pushes the meta-module located at *sPos* into the robot, and pops a meta-module out of the robot at position *ePos*. There are two preconditions for applying this move: (i) *sPos* is at a leaf node in the dual graph of the starting configuration (i.e., it is attached to only one other meta-module) and *ePos* is a leaf node in the dual graph of the ending configuration, and (ii) there is an orthogonal path through the robot starting at *sPos* and ending at *ePos*, with k orthogonal turns (see Figure 2). This move performs an "inchworm" move between successive turns. Thus the contracted "mass" of *sPos* is transferred between turns using $O(1)$ motions.

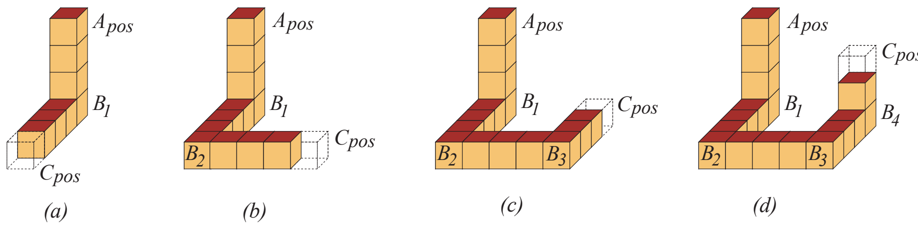

Figure 2: Examples of TUNNEL(A_{pos}, C_{pos}) with orthogonal turns at B_i, $i = 1, 2, 3, 4$. (a) 1-TUNNEL (b) 2-TUNNEL (c) 3-TUNNEL (d) 4-TUNNEL.

In addition to the SLIDE and k-TUNNEL moves, meta-modules can also attach to and detach from adjacent meta-modules. All these moves were explored in [1] for $4 \times 4 \times 4$ meta-modules in the expanded atom model and in [2] for $2 \times 2 \times 2$ meta-modules in the contracted atom model. In the *expanded* (*contracted*) atom model, the state of the meta-module atoms when not performing a move is with all faces expanded (contracted). The appendix illustrates sequences of atomic operations implementing SLIDE and k-TUNNEL for $2 \times 2 \times 2$ meta-modules in the expanded atom model, and we offer a new version of these operations for $2 \times 2 \times 2$ meta-modules in the contracted atom model. Our implementations avoid exchanging atoms among the meta-modules.

As for the complexity, attaching and detaching is done in $O(1)$ parallel steps using $O(1)$ atomic operations. The SLIDE operation is also implemented in $O(1)$ parallel steps using $O(1)$ atomic operations, no matter how many meta-modules are carried in the move. The k-TUNNEL operation is implemented in $O(k)$ parallel steps using $O(k)$ atomic operations, as long as no meta-modules are attached along the path between consecutive turns. Our algorithms ensure this property and only have the need for $k \leq 4$.

2.2. Centralized and Distributed Complexity

We consider both centralized and distributed models of computation. In the centralized model (described in Section 3), computation is performed only by a central processing unit in order to determine the sequence of reconfiguration moves for each meta-module. In Section 4 we briefly discuss how to adapt our algorithms to a synchronous distributed model. While this model does not depend on a central processor, it assumes the existence of a clock, used to synchronize the meta-module moves; each meta-module performs local computations to determine the sequence of moves it needs to perform synchronously.

In this paper we do not address the issue of reducing the computation time; however, we observe that straightforward implementations of our centralized algorithms require $O(n^2)$ computation time. The amount of computation performed by each meta-module in the distributed implementations is $O(n)$. Communication time in both models depends on whether information can be broadcasted to all atoms simultaneously, or if information must propagate through the network of atoms. Since a total of $O(n)$ information must be communicated, this takes $O(n)$ time if broadcasted and $O(n^2)$ time if propagated.

3. Centralized Reconfiguration

In this section we present an algorithm that reconfigures any given source robot, S, into any given target robot, T, where S and T are each a connected set of m meta-modules composed of $n = 8m$ atoms. In Section 3.1, we describe the algorithm for reconfiguring 2D robots which consist of a single layer of meta-modules (all at the same discrete z coordinate in 3D). We then generalize this to 3D robots (Section 3.2).

3.1. Centralized Reconfiguration in 2D

The main idea behind the algorithm is to transform the source robot S into the *common comb* configuration which is defined in terms of both S and T. Then, by executing in reverse the meta-module moves of this algorithm for T, we can transform the common comb into T. Transforming S into the common comb uses an intermediate step in which S is reconfigured into a (regular) *comb*.

3.1.1. 2D Robot to 2D Comb.

In a comb configuration, the meta-modules form a type of histogram polygon [8]. Specifically, the meta-modules are arranged in adjacent columns, with the bottom meta-module of each column in a common row (see Figure 3e). This common row is called the *handle*; the columns of meta-modules extending upward from the handle are called *teeth*.

Initially, the algorithm designates the row containing the topmost meta-modules of S as the *wall* (see Figure 3a). We view the wall as infinite in length. The wall sweeps over the entire robot, moving down one row in each step. By having certain meta-modules slide downward with the wall, the teeth of the comb emerge above the wall. We call this process "combing" the robot. In what follows we will refer to the row of meta-modules immediately above (below) the wall as w^+ (w^-).

Algorithm 1 outlines the combing process. After initializing the wall in Step 1, the loop in line 2 slides the wall down row by row. In each iteration, Step 2.1 labels each wall meta-module as *stationary* (S) if it has a meta-module adjacent below and *moving* (M) otherwise (see Figure 3). Intuitively, moving meta-modules will move downward to occupy the gap below. Step 2.2 identifies *moving wall components*, which are maximal sequences of adjacent moving wall meta-modules. In Figure 3b for example, there are three moving wall components consisting of the 1^{st}, $3^{rd} - 6^{th}$, and 8^{th} wall meta-modules. A moving wall

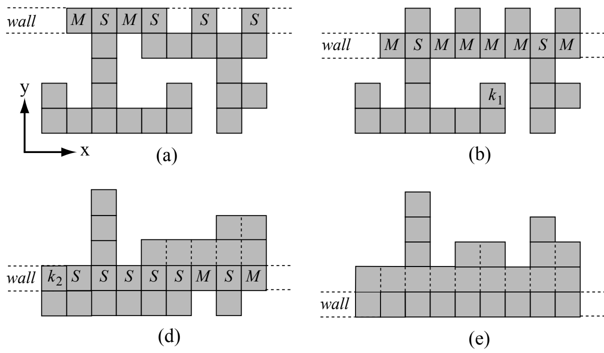

Figure 3: The initial configuration is converted into a comb as it is swept by the wall.

component will always have a stationary meta-module adjacent to one or both ends, for otherwise it would be disconnected from the rest of the robot.

Step 2.3 moves the wall down by one meta-module row. The moving components and the attached teeth move down with the wall. This is accomplished by having each moving wall meta-module adjacent to a stationary meta-module perform a Slide(y^-) move, thus moving itself one row below w.r.t. the adjacent stationary wall meta-module. Figures 3a-3e show the robot configuration after successive moving wall steps.

A series of attach and detach operations in Step 2.4 prepares the robot for the next iteration. First, the end meta-modules of the moved components attach on the left and right to any newly adjacent meta-modules (if not already attached). For example, the meta-module that moves adjacent to k_2 from Figure 3c to 3d will need to attach to k_2. Then each stationary meta-module (now in row w^+) detaches itself from any adjacent meta-modules to its left and right. By doing this, the comb's teeth (which are extending upward from the wall) are disconnected from one another; their only connection to the rest of the robot is through the wall meta-modules at their bases. See Figures 3c-3e where detached meta-modules are separated by dotted lines. The reason for disconnecting the teeth is that in Step 2.3, teeth resting on moving meta-modules get pulled downward while teeth resting on stationary meta-modules stay in place. By disconnecting the teeth, they can move past each other. Finally, all meta-modules in w^- that are now adjacent to a wall meta-module in the y^+ direction attach to these wall meta-modules. Such a situation is illustrated in Figure 3b and 3c, where the meta-module marked k_1 becomes adjacent to a wall meta-module after the sliding step.

Lemma 1. *During* 2D-Combing *a connected robot configuration stays connected.*

Proof. We prove inductively that after each iteration of the loop in line 2 of Algorithm 1, the robot is connected and all adjacent meta-modules in or below the wall are attached. The claim is trivially true after zero iterations, and we

Algorithm 1 2D-COMBING(S)

1.Set wall to row containing topmost meta-modules of S.
2.**While** there are meta-modules below the wall **do**
  2.1 Label wall meta-modules moving or stationary.
  2.2 Identify moving wall components.
  2.3 Move wall one row lower, carrying moving components and attached teeth.
  2.4 Adjust meta-module attachments
    2.4.1 Attach moving components to meta-modules newly adjacent to left (x^-) and right (x^+).
    2.4.2 Detach meta-modules new in w^+ from meta-modules adjacent to left (x^-) and right (x^+).
    2.4.3 Attach meta-modules in w^- to wall meta-modules newly adjacent above (y^+).

assume inductively that it is true after the ith iteration. We now show that it is true after the $(i+1)$st iteration. At the beginning of the $(i+1)$st iteration, consider any maximal moving component in the wall. Let m_l and m_r be its left and right end meta-modules, and let $\mathcal{M}$ be the collection of meta-modules consisting of the moving component plus meta-modules comprising teeth resting on top of it. Since there are no meta-modules adjacent below $\mathcal{M}$ and the teeth in $\mathcal{M}$ are attached only to the moving component at their base, one or both of m_l and m_r must be adjacent to a stationary meta-module in the wall, or else $\mathcal{M}$ is disconnected from the rest of the robot. W.l.o.g. assume, that both m_l and m_r are adjacent to stationary meta-modules, call them s_l and s_r. Let s'_l and s'_r be the adjacent meta-modules below s_l and s_r. In Step 2.3 the moving component slides down, resulting in attachments (s_l, m_l) and (s_r, m_r) being replaced by the attachments (s'_l, m_l) and (s'_r, m_r). Any two meta-modules in the dual graph connected by a path that included edge (s_l, m_l) before the component moved, are still connected via the same path but with (s_l, m_l) replaced by attachments (s_l, s'_l) and (s'_l, m_l). Therefore, the robot S remains connected after the $(i+1)$st move. Step 2.4 in the algorithm ensures that any newly adjacent meta-modules in and below the wall are attached to one another after the $(i+1)$st move. □

Lemma 2. *A 2D robot can transform into its comb configuration in place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

Proof. Clearly, during reconfiguration the robot stays within the bounding box of the source robot. For each of the $O(m)$ iterations, the algorithm performs one parallel set of meta-module SLIDE operations and three parallel sets of attachment operations, which is $O(m) = O(n)$ parallel steps. We now consider the total number of atomic operations performed. For each stationary meta-module that emerges above the wall, there are at most 2 moving meta-modules that slid past it, one on each side. At most m stationary meta-modules emerge above the wall, so the total number of SLIDE operations is bounded by $2m$. Since a meta-module is in w^- at most once and enters the wall and w^+ at most once, the number of meta-module attach and detach operations done in Step 2.4 is $O(m)$. The SLIDE and attach/detach operations require $O(1)$ atomic operations, making the total number of atomic operations performed $O(m) = O(n)$. □

3.1.2. 2D Comb to 2D Common Comb.

For two combs C_S and C_T, this section describes an algorithm to reconfigure C_S into the *common comb* C_{ST}, an intermediate configuration defined in terms of both C_S and C_T.

Let h_S and h_T be the number of meta-modules in the handles of C_S and C_T, and let $h = \max(h_S, h_T)$. Let $S_1, S_2, \ldots, S_h$ denote the teeth of C_S. If $h_S < h_T$, then let $S_{h_S+1}, \ldots, S_h$ be simply "empty teeth." $|S_i|$ is the number of meta-modules on top of the handle meta-module in tooth S_i; it does not count the handle meta-module. We will represent meta-modules by their "coordinates" in the lattice. When referring to meta-modules by their coordinates, we will assume the comb's leftmost handle meta-module is at $(1, 1)$. So the set $\{(i, j) \mid 2 \leq j \leq |S_i|+1\}$ is the set of meta-modules in tooth S_i. All terms are defined analogously for comb C_T and for comb C_U, whose description follows.

Let C_U be a comb that is the union of C_S and C_T in the sense that the length of C_U's handle is h and its ith tooth has length $\max(|S_i|, |T_i|)$, $1 \leq i \leq h$. The common comb C_{ST} is a subset of C_U consisting of its h handle meta-modules and a 'right-fill' of the $m - h$ teeth meta-modules into the shell defined by C_U. For example, Figs. 4a and 4b show C_S and C_T. In Figure 4d, C_U consists of all the shaded and unshaded meta-modules; the common comb is all the shaded boxes.

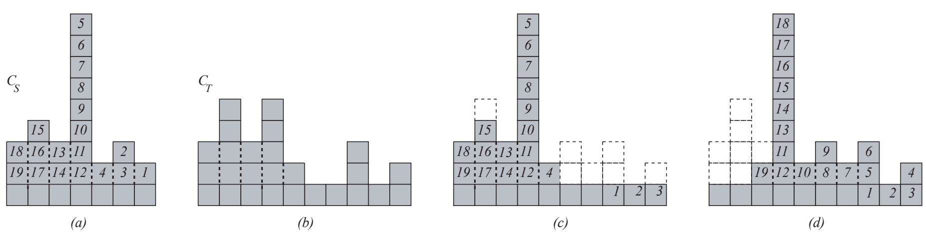

Figure 4: (a) C_S, with meta-modules labeled in reverse lexicographical order. (b) C_T. (c) Shaded meta-modules are C_S after extending its handle's length to match C_U. C_U consists of all shaded and unshaded boxes. Labels indicate which meta-modules moved to form the handle. (d) Shaded meta-modules form the common comb for C_S and C_T.

Algorithm 2 describes in detail the process of converting C_S to the common comb. Step 1 initializes queue O with the teeth meta-modules of C_S in reverse lexicographical order on their coordinates. (See the labeled ordering in Figure 4a.) This is the order in which teeth will be moved to fill in for missing meta-modules in the common comb. We assume O supports the operations $O.dequeue()$ and $O.size()$, where $O.dequeue()$ removes and returns the front item in O and $O.size()$ returns the number of items in O. Step 2 lengthens

C_S's handle so that it contains h meta-modules, moving meta-modules from O to the handle using 1-Tunnel operations. Figure 4c shows the results of Step 2.

Once the handle is of proper length, C_S's teeth are lengthened to match the lengths of C_U's teeth, starting with the rightmost tooth. Since C_U is the union of C_S and C_T, each tooth S_i of C_S is either the same length as the corresponding tooth in C_U, or it is shorter. A key invariant of the algorithm is that, at the beginning of an iteration in Step 3, O contains exactly those meta-modules in teeth $S_1, \ldots, S_i$ of C_S. This is certainly true in the first iteration when $i = h$, and can be easily shown to be true inductively for all i. Therefore, at the start of an iteration, if $|S_i| > 0$ then the next $|S_i|$ meta-modules in O are exactly the teeth meta-modules in S_i. These meta-modules are already in their final locations, and so they are just removed from O (Loop 3.1). Loop 3.2 then moves the next $|U_i| - |S_i|$ teeth meta-modules in O to tooth S_i using 2-Tunnel operations. Figure 4d shows the resulting common comb.

Observe that in Loop 3.2, tooth $oPos$ is always the top meta-module of the first non-empty tooth to the left of tooth S_i. Therefore, the orthogonal path followed in the 2-Tunnel operation goes from $oPos$ down to the handle meta-module at the base of the tooth, through a (possibly length 0) section of the handle containing only empty teeth, and then up to the top of tooth i. No meta-modules are attached between turns along this path, so the 2-Tunnel operation requires only $O(1)$ atomic operations to complete.

Algorithm 2 2D-Comb-To-Common-Comb(C_S, C_U)

---

1. Let O be a queue of the (i, j) coordinates of the teeth meta-modules (i.e., $j > 1$) of C_S, in reverse lexicographical order.
2. If $h_S < h$ then { extend C_S's handle to length h }
   - 2.1 For $i = h_S + 1$ to h
     - 2.1.1 oPos = $O.dequeue()$
     - 2.1.2 In C_S, 1-Tunnel($oPos$,$(i, 1)$)
3. For $i = h$ down to 1 { lengthen teeth of C_S, from right to left }
   - 3.1 For $j = 1$ to $|S_i|$
     - 3.1.1 $O.dequeue()$ { remove meta-modules already in tooth S_i }
   - 3.2 For $j = |S_i| + 1$ to $|U_i|$ { lengthen tooth S_i }
     - 3.2.1 if $O.size() = 0$ then exit
     - 3.2.2 $oPos = O.dequeue()$
     - 3.2.3 In C_S, 2-Tunnel($oPos$,(i, j))

Lemma 3. *A 2D robot can transform into a common comb configuration in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

Proof. The reconfiguration takes place within the union of the bounding boxes of C_S and C_T, which is contained within the union of the bounding boxes of S and T. At most m modules are relocated, each by a 1-Tunnel or 2-Tunnel operation requiring $O(1)$ atomic operations, resulting in $O(m) = O(n)$ parallel steps and atomic operations. □

3.1.3. Overall 2D Reconfiguration Algorithm.

The general algorithm to reconfigure any m meta-module robot S to any other m meta-module robot T consists of four major steps. First S reconfigures into comb C_S, then C_S reconfigures into common comb C_{ST}. Then the reverse moves of the 2D-Comb-To-Common-Comb and 2D-Combing algorithms reconfigure C_{ST} into C_T and then C_T into T.

Theorem 1. *Any 2D source robot can be reconfigured into any 2D target robot in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

3.2. Centralized Reconfiguration in 3D

Analogous to the 2D case, in 3D the source robot S is also transformed into a 3D common comb and then into the target robot T. In transforming to the 3D common comb there are two intermediate configurations, a terrain configuration and a (regular) 3D comb configuration.

3.2.1. Source Robot to 3D Terrain.

We use the 3D analog of the 2D-Combing process, 3D-Combing, to reconfigure S into a 3D terrain. The 3D algorithm is the same as in 2D, except the wall now consists of an entire 2D horizontal layer of meta-modules, initially the topmost single layer of S. See Figure 5. In each iteration of the algorithm, wall meta-modules are labeled as stationary or moving. Analogous to the 2D case, a stationary meta-module is one that has an adjacent meta-module below. Here, a 3D moving wall component is an arbitrarily shaped maximal component of adjacent moving meta-modules within the wall. In Figure 5a for instance, the wall is in the initial position and contains one single F-shaped moving component. When the wall moves down a layer, the moving components slide past

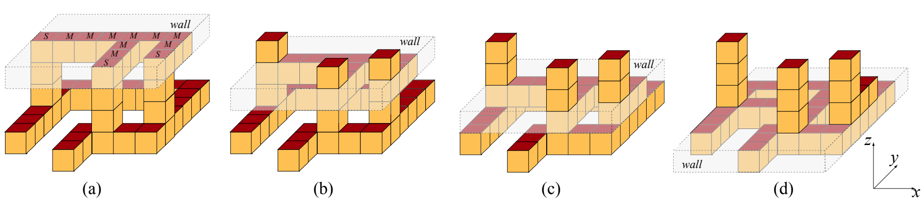

Figure 5: The 3D-Combing algorithm. (a) Meta-modules labeled M form one F-shaped connected component. (b, c, d) Robot configuration after (1, 2, 3) algorithm iterations. (d) Final terrain configuration.

the stationary meta-modules (using a Slide(z^-) move). The final result is that all meta-modules of S having the same (x, y) coordinates are grouped together to form a contiguous tower of meta-modules. These towers extend in the z^+ direction, rest on an arbitrarily-shaped, connected base layer (in the xy-plane), and are attached only to the base layer.

Lemma 4. *A 3D robot can transform into a 3D terrain in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

3.2.2. 3D Terrain to 3D Comb.

A 3D Terrain I is reconfigured into a 3D comb by applying the 2D-COMBING algorithm of Section 3.1.1 to its base layer, thus reconfiguring the base layer into a 2D comb. As the base meta-modules move during the reconfiguration, they carry along the towers resting on top. If $B(I)$ is the base of I, then a call to 2D-COMBING($B(I)$) using the SLIDE operation that carries towers (see Figure 1c) accomplishes this. After this second combing pass, the resulting 3D comb robot consists of a 2D comb in the xy-plane (call this the xy-comb), and each tooth and its handle module in the xy-comb form the handle of a comb with teeth extending up in the z direction (call these the z-combs). We immediately have the following result.

Lemma 5. *A 3D terrain can transform into a 3D comb in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

3.2.3. 3D Comb to 3D Common Comb.

Given two 3D combs C_S and C_T, this section describes an algorithm to reconfigure C_S into the 3D common comb C_{ST} determined by C_S and C_T. Let s (t) be the number of z-combs in C_S (C_T); equivalently, s (t) is the handle length of C_S's (C_T's) xy-comb. We assume C_S (C_T) is positioned with the handle of its xy-comb starting at lattice coordinates $(1,1,1)$ and extending to $(s,1,1)$ ($(t,1,1)$). Let C_S^i be the z-comb of C_S in lattice position i, let S_j^i be the jth tooth of C_S^i, and let $|S_j^i|$ be the number of teeth meta-modules in tooth S_j^i (not counting the handle module at its base). Let h_S^i be the length of C_S^i's handle. All terms are defined analogously for combs C_T and C_U.

As in 2D, comb C_U is the union of C_S and C_T. Let u be the handle length of C_U's xy-comb. The common comb is a subset of C_U consisting of the u handle meta-modules in its xy-comb and its rightmost $m-u$ meta-modules. More precisely, for each z-comb C_U^i, $i = u \dots 1$, append to a list I the handle meta-modules $(i,2,1)$ to $(i,h_U^i,1)$ of C_U^i, followed by the teeth meta-modules of C_U^i in descending order on their y coordinate (primary key) and increasing order on their z coordinate (secondary key). The first $m-u$ meta-modules of I are in the common comb.

Algorithm 3 describes in detail the process of converting C_S to the common comb. In Step 1, the algorithm converts each z-comb C_S^i to the 2D common comb determined by $C_U^i = C_S^i \cup C_T^i$ using Algorithm 2. Figure 6a shows example results from Step 1. Since C_S^i and C_T^i may not contain the same number of meta-modules, there may not be enough meta-modules in C_S^i to fill the entire handle of C_U^i, in which case C_S^i will contain only a portion of the handle that starts with module $(i,1,1)$.

Step 2 creates a queue, O, of meta-modules, in the order in which they will be used to fill meta-modules of C_U. Step 3 extends the length of C_S's xy-comb handle so that it matches the length of C_U's xy-comb handle. Figure 6b shows the results of this step. The order of the meta-modules in O ensures that each leg of the k-TUNNEL path is unattached to other meta-modules, thus allowing the k-TUNNEL move to be performed in $O(1)$ time. In Step 4, the teeth of each

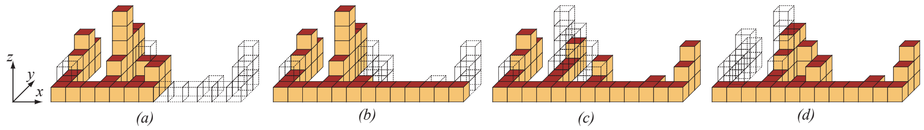

Figure 6: (a) Solid meta-modules are C_S after each z-comb is converted to a common comb. C_U consists of the solid and the wireframe boxes. (b) C_S after extending its xy-comb handle to match that of C_U. (c) C_S during the execution of Step 4.3 of Algorithm 3, as it lengthens the teeth of C_S^7 by tunneling meta-modules from C_S^4. (d) The 3D common comb (solid boxes only).

z-comb in C_S are lengthened to match the lengths of the corresponding teeth in C_U. As in 2D, an important invariant is that, at the beginning of each iteration i of Step 4, O contains exactly the teeth and handle meta-modules in combs $C_S^1, \ldots, C_S^i$ (with the exception of those meta-modules in C_S's xy-comb handle, which stay in place throughout this step). Step 4.1 removes from O those meta-modules that are already in C_S^i. Step 4.2 extends C_S^i's handle such that its length matches that of C_U^i. Step 4.3 lengthens short teeth of C_S^i. Again, the order of the meta-modules in O ensures that each k-Tunnel operation follows a path whose segments are not attached to any other meta-modules, allowing $O(1)$ k-Tunnel moves. A stage of Step 4 is illustrated in Figure 6c, with Figure 6d showing the resulting 3D common comb (solid meta-modules).

Lemma 6. *A 3D robot can transform into a common comb configuration in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

3.2.4. Overall 3D Reconfiguration Algorithm.

The general algorithm to reconfigure any 3D m meta-module robot S to any 3D m meta-module target robot T consists of six stages: S reconfigures into 3D terrain I_S, then I_S reconfigures into 3D comb C_S, then C_S reconfigures into common comb C_{ST}, and finally the reverse moves reconfigure C_{ST} into C_T, C_T into I_T, and then I_T into T.

Theorem 2. *Any source robot can be reconfigured into any target robot in-place in $O(n)$ parallel steps and a total of $O(n)$ atomic operations.*

We have focused on reducing the mechanical operations performed by the atoms because they dominate the reconfiguration time. However, we comment briefly on the time required by the centralized processor to compute the sequence of meta-module moves. A straightforward simulation of the 3D-Combing algorithm takes $O(m)$ time to slide the wall down one row, giving an overall running time of $O(m^2)$. The Terrain-To-Comb algorithm has the same time behavior as the 2D-Combing algorithm, which also requires $O(m^2)$ time. The 3D-Comb-To-Common-Comb algorithm requires $O(m)$ time, if counting sort is used to order the meta-modules in the queue O by their lattice coordinates, and counts such as $|C_S^i|$ are initially determined and then updated each time a meta-module is relocated via a k-Tunnel operation. Therefore, the overall processing time required by the centralized processor is $O(m^2)$, which is $O(n^2)$. In

Algorithm 3 3D-Comb-To-Common-Comb Algorithm(C_S, C_U)

1. For $i = 1 \dots s$
    1.1 2D-Comb-To-Common-Comb(C_S^i, C_U^i)
        (with combs parallel to the yz plane)
2. Let O be an empty queue
    For $i = s$ down to 1
    2.1 Append to O the teeth meta-modules of C_S^i, ordered by increasing y (primary key) and decreasing z (secondary key)
    2.2 Append to O all handle meta-modules of C_S^i except for module $(i, 1, 1)$, ordered by decreasing y
3. If $s < u$ then { extend the handle of C_S's xy-comb to length u }
    3.1 For $i = s + 1$ to u
        $oPos = O.dequeue()$
        In C_S, k-Tunnel($oPos$, $(i, 1, 1)$), for $k \in \{1, 2\}$
4. For $i = u$ down to 1 { fill in missing meta-modules of each z-comb }
    4.1 For $j = 1$ to $|C_S^i| - 1$
        $O.dequeue()$ { remove meta-modules already in C_S^i }
    4.2 For $j = h_S^i + 1$ to h_U^i { lengthen handle of C_S^i }
        If $(O.size() == 0)$ exit
        $oPos = O.dequeue()$
        In C_S, k-Tunnel($oPos$, $(i, j, 1)$), for $k \in \{2, 3\}$
    4.3 For $j = h_S^i$ down to 1 { lengthen short teeth of C_S^i }
        For $k = |S_j^i| + 1$ to $|U_j^i|$
            If $(O.size() = 0)$ exit
            $oPos = O.dequeue()$
            In C_S, k-Tunnel($oPos$, (i, j, k)), for $k \in \{3, 4\}$

the distributed algorithm that follows, this reduces to $O(n)$ (parallel) processing time.

4. Distributed Implementation

Our centralized algorithms can be executed by the meta-modules in a synchronous, distributed fashion. The implementation must be synchronous since both the SLIDE and k-TUNNEL moves require strict coordination of motion among the atoms in order to prevent collisions and disconnection of the robot. For example, the two end meta-modules of a moving component in the 2D-COMBING algorithm must perform their slides at the same time. To synchronize the operations, we assume each atom/meta-module can count clock ticks modulo k, for any $k \in \mathbb{N}$.

The COMBING algorithm can be easily adapted to the synchronous distributed model. During an initialization phase, each meta-module is sent its starting (x, y, z) location and the starting position of the wall. Thereafter, each meta-module can determine its next move in $O(1)$ time by keeping track of elapsed clock ticks to determine the position of the wall, using information on its current state (moving or stationary), and polling adjacent meta-modules on their state. For example, each meta-module can determine its own moving or stationary label by just checking if it is attached to a module below.

For the reverse of the COMBING algorithm taking C_T to T, if a meta-module were to simulate the entire forward combing algorithm to determine the sequence of moves it will run in reverse, its overall processing time would be $O(n^2)$. But we describe here a distributed reverse combing algorithm that requires each meta-module to do only $O(n)$ processing, albeit with some stronger requirements. It assumes that each meta-module's location in C_T and the final configuration is communicated to every meta-module. In addition, each meta-module requires a more powerful processor on board. Specifically, we require that each meta-module can store information of size $O(n)$ and can run an algorithm of complexity $O(n)$ in $O(n)$ time.

We will describe the algorithm for 2D; for 3D it is analogous. Using the final configuration T and its (x, y) location in C_T, each meta-module first determines in $O(n)$ time where its final location in T will be. Its final location has coordinates (x, y'), where $y' \geq y$. The wall initially is set to the first row of C_T and slides upwards. For each movement of the wall, the wall meta-modules label themselves as moving or stationary. A wall meta-module is moving if its current y coordinate is less than its final y' coordinate. Otherwise, it is stationary (and is in its final location). Next, moving components are identified. At least one end meta-module of each moving component is adjacent to a stationary module, which in turn has a meta-module resting on top of it, for otherwise the final configuration T would not be connected. When a meta-module is labeled a moving meta-module for the first time, it detaches from the meta-module adjacent below it (if any). Then the end meta-modules of the moving components slide their component and any meta-modules resting on top of it up one row. In preparation for the next slide, all meta-modules that entered the wall for the first time attach on the left and right to any adjacent meta-modules. A minor issue arises in having the meta-modules keep track of their (x, y) locations as the moving components slide up, since meta-modules resting on top of moving components need to know that they moved up a row. It is straighforward though to have each meta-module use C_T and T to determine in $O(n)$ time the clock tick in which the wall will reach it and what its y coordinate will be on that clock tick. Like the forward COMBING algorithm, the reverse algorithm is in-place and runs in $O(n)$ parallel steps and performs a total of $O(n)$ atomic operations.

The COMB-TO-COMMON-COMB algorithms can also be distributed, albeit with similar requirements as the distributed reverse combing algorithm. First, the initial and final configurations S and T must be communicated to each meta-module. Then each meta-module must simulate the COMB-TO-COMMON-COMB algorithm to precompute which operations it will perform on each clock strike, since local information alone is not enough to determine a meta-module's next operation. For example, meta-modules at the turn locations in the k-TUNNEL operations must determine when they will be involved in such an operation in order to coordinate their actions. Distributing the reverse of the COMB-TO-COMMON-COMB algorithm to take C_{ST} to C_T is done similarly and thus has the same requirements as the forward COMB-TO-COMMON-COMB algorithm.

The total processing time needed to determine the atomic operations is reduced to $O(n)$ parallel time in the distributed implementation. In the forward COMBING algorithms, each meta-module does $O(1)$ computations in parallel for each of the $O(m) = O(n)$ times the wall moves. Similarly, the reverse COMBING algorithms require $O(n)$ parallel time, as outlined above. In the forward and reverse COMB-TO-COMMON-COMB algorithms, in parallel each atom simulates the algorithm to precompute its moves, taking $O(n)$ parallel time.

5. Conclusions and Open Problems

Here we presented an algorithm for reconfiguring a cube-style robot in $O(n)$ parallel moves and atomic operations, improving upon the previous best algorithms that required both $O(n^2)$ parallel moves and atomic operations. Subsequent to the short version of this result appearing in [4], there have been two significant new advances: an algorithm that reconfigures in $O(n)$ parallel operations while taking into account the physical forces associated with mass or inertia during the moves by restricting to constant force per operation [9], and an algorithm for reconfiguring in $O(\log n)$ parallel moves [10]. Problems open for future work include developing distributed algorithms that operate using simple local rules, thus reducing the on-board processing requirements of the atoms. Also of interest are algorithms that minimize the maximum number of moves any one atom must perform, since each atom is limited by its battery power.

Acknowledgments. We thank Thomas Hackl for his suggestion on how to reduce the size of the meta-modules [7]. We thank the other participants of the 2007 *Workshop on Reconfiguration* at the Bellairs Research Institute of McGill University for providing a stimulating research environment.

References

[1] D. Rus, M. Vona, Crystalline robots: Self-reconfiguration with compressible unit modules, Autonomous Robots 10 (1) (2001) 107–124.

[2] S. Vassilvitskii, M. Yim, J. Suh, A complete, local and parallel reconfiguration algorithm for cube style modular robots, in: Proc. of the IEEE Intl. Conf. on Robotics and Automation, 2002, pp. 117–122.

[3] Z. Butler, D. Rus, Distributed planning and control for modular robots with unit-compressible modules, The Intl. Journal of Robotics Research 22 (9) (2003) 699–715. doi:10.1177/02783649030229002.

[4] G. Aloupis, S. Collette, M. Damian, E. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, S. Wuhrer, Linear reconfiguration of cube-style modular robots, in: Proc. 18th International Symposium on Algorithms and Computation, Vol. 4835 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 208–219.

[5] Z. Butler, R. Fitch, D. Rus, Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting, IEEE/ASME Trans. on Mechatronics 7 (4) (2002) 418–430.

[6] J. Suh, S. Homans, M. Yim, Telecubes: Mechanical design of a module for self-reconfigurable robotics, in: Proc. of the IEEE Intl. Conf. on Robotics and Automation, 2002, pp. 4095–4101.

[7] T. Hackl, Personal communication (2007).

[8] E. M. Arkin, M. A. Bender, J. S. B. Mitchell, V. Polishchuk, The snowblower problem, in: S. Akella, N. M. Amato, W. H. Huang, B. Mishra (Eds.), WAFR, Vol. 47 of Springer Tracts in Advanced Robotics, Springer, 2006, pp. 219–234.

[9] G. Aloupis, S. Collette, M. Damian, E. Demaine, D. El-Khechen, R. Flatland, S. Langerman, J. O'Rourke, V. Pinciu, S. Ramaswami, V. Sacristán, S. Wuhrer, Realistic reconfiguration of crystalline (and telecube) robots, in: Proc. 8th International Workshop on the Algorithmic Foundations of Robotics, To appear 2008.

[10] G. Aloupis, S. Collette, E. Demaine, S. Langerman, V. Sacristán, S. Wuhrer, Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves, in: Proc. 19th International Symposium on Algorithms and Computation, To appear 2008.

Appendix

This appendix shows how to achieve the SLIDE and k-TUNNEL moves in both the expanded and the contracted atom models, using meta-modules of minimum size. In the expanded (contracted) model, the atoms have their faces expanded (contracted) except when they are involved in a meta-module operation. Both models have been considered in the robotics community, and one may be preferred over the other depending on whether space is restricted (use the contracted model) or a larger robot is needed (use the expanded model). Previously, meta-modules of size $4 \times 4 \times 4$ were thought to be required to perform these moves in the expanded model [1]. Here we show that $2 \times 2 \times 2$ meta-modules suffice [7]. For both models, the sequences of atom operations we provide for the k-TUNNEL move avoid exchanging atoms among meta-modules.

SLIDE

The following figures show the first steps in an example SLIDE operation applied to the top meta-module (dark gray) in the expanded model (Figure 7) and in the contracted model (Figure 8). In both cases, the result is that the top meta-module slides one atom to the left; repeating this sequence of steps one more time completes the SLIDE move.

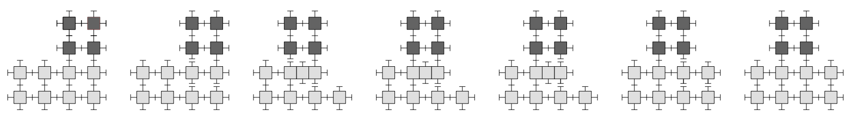

Figure 7: Expanded robot. First steps of SLIDE applied to the top meta-module. Only one layer shown.

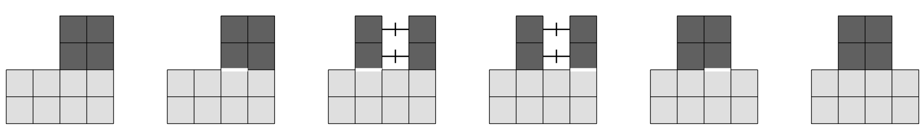

Figure 8: Contracted robot. First steps of SLIDE applied to the top meta-module. Only one layer shown.

Figures 9 and 10 show how the sliding meta-module can carry other meta-modules with it.

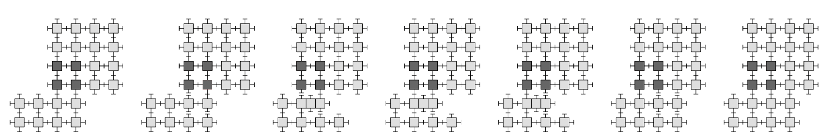

Figure 9: Expanded robot. Carrying meta-modules in a SLIDE move. Only one layer shown.

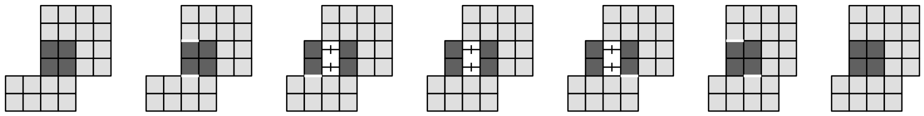

Figure 10: Contracted robot. Carrying meta-modules in a Slide move. Only one layer shown.

Connectedness of the robot. Notice that sliding one meta-module will not disconnect the robot as long as the sliding meta-module (dark gray in the previous figures) is only attached to the substrate meta-module (colored light gray) with respect to which it will slide. When several modules are to slide simultaneously, possibly carrying some other meta-modules with them, both the sliding and the carried meta-modules need to be only attached to meta-modules sliding or being carried in the same direction.

Complexity of the move. As shown in Figures 9 and 10, both the number of parallel steps and the number of atomic operations (contract, expand, attach, detach) needed to perform a Slide move is constant, no matter how many meta-modules are carried in the move.

k-Tunnel

Figures 11 and 12 show the 1-Tunnel($(x, y{+}1)$, $(x{+}1, y)$) atomic operations in the expanded and contracted models. Figures 13 and 14 show selected steps of 1-Tunnel($(x, y+3)$, $(x+3, y)$) in both models.

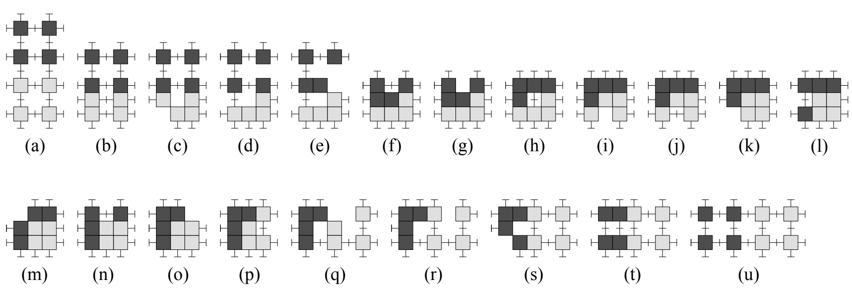

Figure 11: Expanded robot. 1-Tunnel($(x, y+1)$, $(x+1, y)$). Only one layer shown. Attachments and detachments are not shown.

Connectedness of the robot. In all our algorithms, we have no modules attached along the path between the meta-modules where the path turns. So tunneling a meta-module along a path can be achieved without worry about disconnecting the robot.

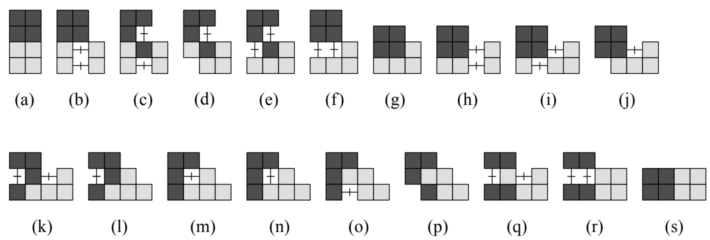

Figure 12: Contracted robot. Example of 1-Tunnel$((x, y+1), (x+1, y))$. Only one layer shown. Attachments and detachments are not shown.

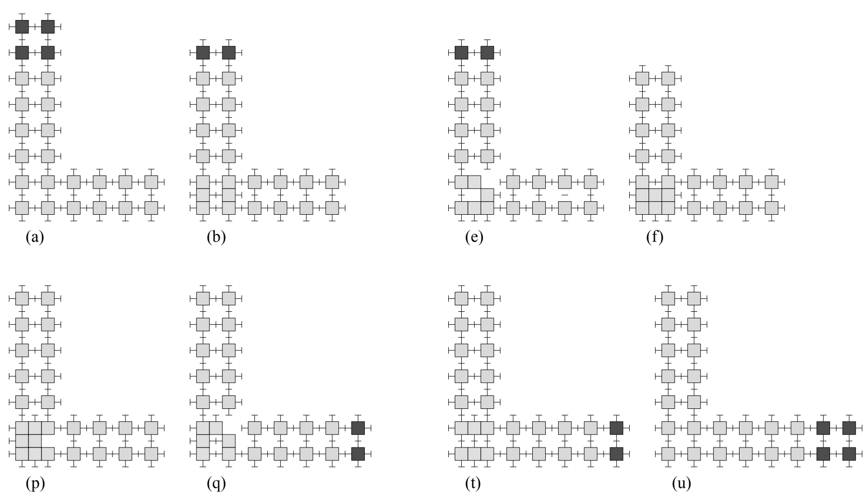

Figure 13: Expanded robot. Selected steps of 1-Tunnel$((x, y+3), (x+3, y))$. Labels refer to Figure 11.

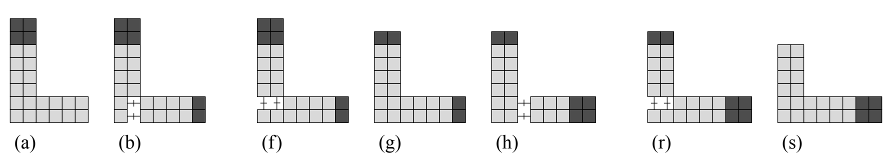

Figure 14: Contracted robot. Selected steps of 1-Tunnel$((x, y+3), (x+3, y))$. Labels refer to Figure 12.

Complexity of the move. k-Tunnel is implemented in $O(k)$ parallel steps using $O(k)$ atomic operations, as long as there are no meta-modules attached along the paths between consecutive turns, as is the case in our algorithms here. When there are attached meta-modules, k-Tunnel can still be implemented in $O(k)$ parallel steps, but the number of atomic operations required is proportional to the number of turns plus the number of meta-modules attached to the legs of the path. This is because each attached meta-module must be partially detached to allow the pushing along the legs of the path.