

# Linear Reconfiguration of Cube-Style Modular Robots

Greg Aloupis<sup>1</sup>, Sébastien Collette<sup>2</sup>, Mirela Damian<sup>3</sup>, Erik D. Demaine<sup>4</sup>, Robin Flatland<sup>5</sup>, Stefan Langerman<sup>6</sup>, Joseph O'Rourke<sup>7</sup>, Suneeta Ramaswami<sup>8</sup>, Vera Sacristán<sup>9</sup>\*, and Stefanie Wuhrer<sup>10</sup>

<sup>1</sup> Université Libre de Bruxelles, Belgique, [greg@scs.carleton.ca](mailto:greg@scs.carleton.ca)

<sup>2</sup> Université Libre de Bruxelles, Belgique, [sebastien.collette@ulb.ac.be](mailto:sebastien.collette@ulb.ac.be)

<sup>3</sup> Villanova University, Villanova, USA, [mirela.damian@villanova.edu](mailto:mirela.damian@villanova.edu)

<sup>4</sup> Massachusetts Institute of Technology, Cambridge, USA, [edemaine@mit.edu](mailto:edemaine@mit.edu)

<sup>5</sup> Siena College, Loudonville, N.Y., USA, [flatland@siena.edu](mailto:flatland@siena.edu)

<sup>6</sup> Université Libre de Bruxelles, Belgique, [stefan.langerman@ulb.ac.be](mailto:stefan.langerman@ulb.ac.be)

<sup>7</sup> Smith College, Northampton, USA, [orourke@cs.smith.edu](mailto:orourke@cs.smith.edu)

<sup>8</sup> Rutgers University, Camden, USA, [rsuneeta@camden.rutgers.edu](mailto:rsuneeta@camden.rutgers.edu)

<sup>9</sup> Universitat Politècnica de Catalunya, Barcelona, Spain, [vera.sacristan@upc.edu](mailto:vera.sacristan@upc.edu)

<sup>10</sup> Carleton University, Ottawa, Canada, [swuhrer@scs.carleton.ca](mailto:swuhrer@scs.carleton.ca)

**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  $O(n)$  atom operations (expand, contract, attach, detach) and is performed in  $O(n)$  parallel steps. This improves on previous reconfiguration algorithms [RV01,VYS02,BR03], 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. <sup>11</sup>

## 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 are comprised of 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 face of an atom is equipped with 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 [BFR02] and telecube atoms [SHY02]. 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 arranged in *meta-modules*, which are groups of  $k \times k \times k$  atoms attached to one another in a cubic shape.

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

\* Partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

<sup>11</sup> This paper was accepted to ISAAC 2007 and will appear in abbreviated form in LNCS. Copyright ©Springer-Verlag, 2007. URL: <http://www.springer.de/comp/lncs/index.html>.

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<sup>12</sup>, reconfigures  $S$  into  $T$  in  $O(n)$  parallel steps and a total of  $O(n)$  atom operations. Our algorithm improves significantly the previously best-known reconfiguration algorithms for cube-style modular robots [RV01,VYS02,BR03], which take  $O(n^2)$  parallel steps as well as  $O(n^2)$  atom 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 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.

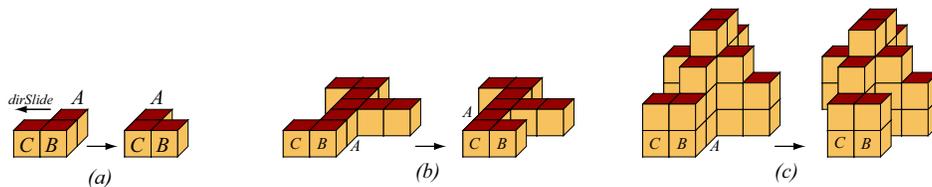
## 2 Preliminaries

### 2.1 Robots as Lattices of Meta-Modules

There exist atom configurations which cannot be reconfigured, e.g. a single row of atoms. But connectedness of the reconfiguration space can be guaranteed for robots composed of *meta-modules* [RV01,VYS02], where a meta-module is a connected set of  $k^3$  atoms arranged in a  $k \times k \times k$  grid. It is obviously desirable that meta-modules be composed of as few atoms as possible. In the reconfiguration algorithms that we propose, meta-modules are of minimum size consisting of a  $2 \times 2 \times 2$  grid of atoms [Hac07,VYS02].

We define two basic meta-module moves (hardware independent) used by our reconfiguration algorithms, similar to the ones described in [VYS02].

**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 Fig. 1a) is adjacent to a meta-module in a direction orthogonal to *dirSlide* ( $B$  in Fig. 1a), which in turn is adjacent to a meta-module in direction *dirSlide* ( $C$  in Fig. 1a) and (ii) the target position for the sliding meta-module is free. This move allows the sliding meta-module to “carry” other attached meta-modules (as in Figure 1b),

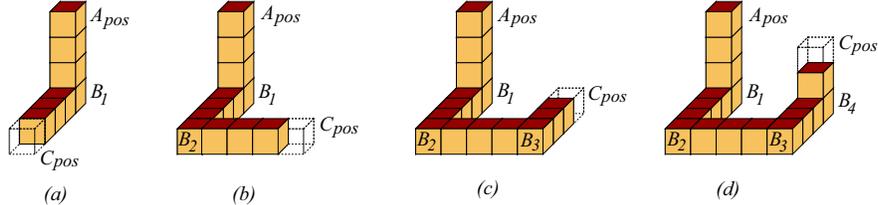


**Fig. 1.** Examples of  $\text{SLIDE}(x^-)$ : (a) Meta-module  $A$  slides alone, (b)  $A$  carries adjacent meta-modules, (c)  $A$  carries towers.

as long as the target position for a carried meta-module is unoccupied and the carried meta-module is only attached to other meta-modules moving simultaneously in the same direction. Similarly, this move allows entire towers of meta-modules sitting on top of the moving meta-modules to be carried along, as shown in Figure 1c.

<sup>12</sup> Throughout the paper,  $n$  refers to the number of robot atoms and  $m$  refers to the number of robot meta-modules, where  $n = 8m$ .

$k$ -TUNNEL( $sPos$ ,  $ePos$ ). Pushes the meta-module located at  $sPos$  (start position) into the robot, and pops a meta-module out of the robot in position  $ePos$  (end position). The preconditions for applying this move are: (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. This move is illustrated in Figure 2 for  $k = 1, 2, 3, 4$ . It performs an “inchworm” move between successive turns. Thus the contracted “mass” of  $sPos$  is transferred between turns using  $O(1)$  motions.



**Fig. 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 the Appendix, we illustrate the sequences of atom operations required to implement SLIDE and  $k$ -TUNNEL for two types of robots: (i) those whose atoms’ natural position is contracted, and (ii) those whose atoms’ natural position is expanded. In both cases, the robot stays connected at all times during a meta-module slide or tunnel move. In addition to these two moves, meta-modules can also attach to and detach from adjacent meta-modules.

As for the complexity, attaching and detaching is done in  $O(1)$  parallel steps using  $O(1)$  atom operations. The SLIDE operation is also implemented in  $O(1)$  parallel steps using  $O(1)$  atom operations, no matter how many meta-modules are carried in the move. The  $k$ -TUNNEL is implemented in  $O(k)$  parallel steps using  $O(k)$  atom 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

As stated previously, our reconfiguration algorithms execute meta-module moves in parallel. Because the mechanical operations are the dominating factor in the reconfiguration time, our focus here is on minimizing the number of parallel moves. To reduce the total power consumption, we are also interested in minimizing the total number of moves performed over all meta-modules.

We consider both centralized and distributed models of computation. The centralized model assumes the existence of a central processing unit that controls the actions of the atoms. In this model, computation is performed only by the central processing unit in order to determine the sequence of reconfiguration moves for each meta-module. In this paper we do not address the issue of reducing the computation time; however, we observe that straightforward implementations of our centralized algorithms (described in Section 3) require  $O(n^2)$  computation time. Communication time in the centralized model depends on whether the central unit can broadcast information to all atoms simultaneously, or if information must propagate through the network of atoms. Since a total of  $O(n)$  atom operations must be communicated, this takes  $O(n)$  time if broadcasted and  $O(n^2)$  if propagated.

In section 5 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. In this model, each meta-module performs local computations to determine the

sequence of moves it needs to perform synchronously. The amount of computation performed by each meta-module is  $O(n)$ . Only the initial and final robot configurations need to be communicated, which takes  $O(n)$  time if broadcasted and  $O(n^2)$  if propagated.

### 3 Centralized Reconfiguration

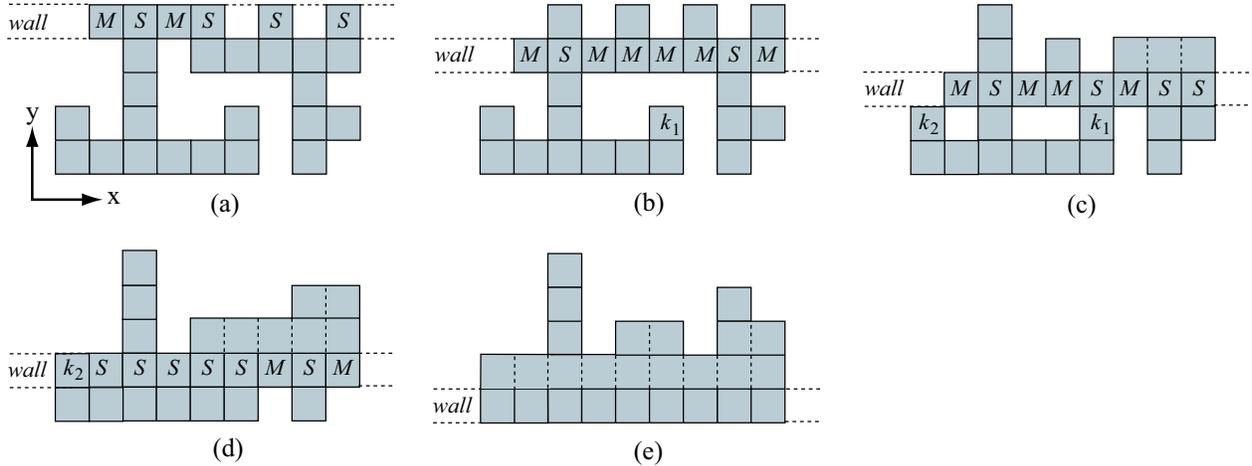
In this section we present an algorithm that reconfigures any given source robot,  $S$ , into any given target robot,  $T$ , where  $S$  and  $T$  are each a connected set of  $m$  meta-modules composed of  $n = 8m$  atoms. We describe the algorithm first for reconfiguring 2D robots which consist of a single layer of meta-modules (Section 3.1). 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  $T$ . In transforming  $S$  into the common comb, there is an intermediate step in which  $S$  is reconfigured into a (regular) *comb*. Section 3.1 describes the algorithm for converting a robot to a comb, and Section 3.1 describes how to convert from the comb to the common comb.

**2D Robot to 2D Comb** In a comb configuration, the meta-modules form a type of histogram polygon [ABMP06]. 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  $w^+$  ( $w^-$ ).



**Fig. 3.** The initial configuration is converted into a comb as it is swept by the wall. At each step, meta-modules in the wall are labeled moving (marked with an M) or stationary (marked with an S).

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<sup>st</sup>, 3<sup>rd</sup> – 6<sup>th</sup>, and 8<sup>th</sup> wall meta-modules. As will become clear in Lemma 1, a moving 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 teeth attached to them move down with the wall. This is done by having each moving wall meta-module adjacent to a stationary meta-module perform a  $\text{SLIDE}(y^-)$  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 attach to this wall meta-module. 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.

---

**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 along moving wall components and their 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 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 ( $y^+$ ).
- 

**Lemma 1.** *The robot configuration forms one connected component at all times.*

*Proof.* We prove inductively that after the  $i$ th 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 assume inductively that it is true after the  $i$ th iteration. We now show it true after the  $(i + 1)$ st iteration. At the beginning of the  $(i + 1)$ st iteration, consider any identified moving component in the wall. Let  $m_l$  and  $m_r$  be its left and right end meta-modules, and let  $M$  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  $M$  and the teeth in  $M$  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  $M$  is disconnected from the rest of the robot. Wlog, assume 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 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.

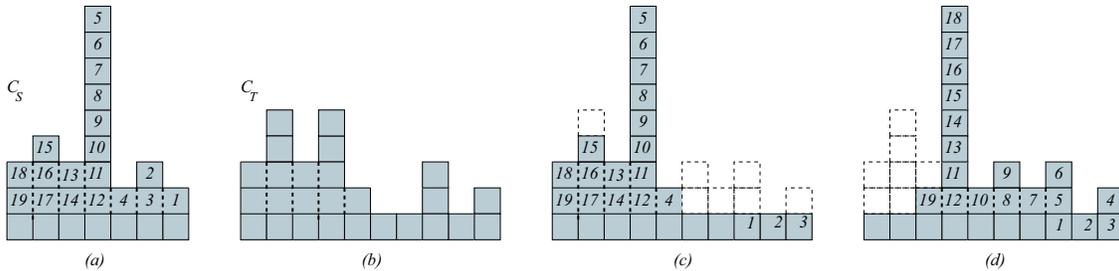
**Lemma 2.** *A 2D robot can transform into its comb configuration in place in  $O(n)$  parallel steps and a total of  $O(n)$  atom operations.*

*Proof.* Clearly the reconfiguration is within the bounding box of the source robot. For each of the  $O(m)$  iterations, it performs one parallel set of meta-module SLIDE operations and three parallel attachment operations, which is  $O(m) = O(n)$  parallel steps. We now consider the total number of atom 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 either 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  $w^+$  and  $w^-$  at most once and enters the wall 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  $O(1)$  atom operations, making the total number of atom operations performed  $O(m) = O(n)$ .

**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*, 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  $S_1, S_2, \dots, S_h$  denote the teeth of  $C_S$ . If  $h_S < h_T$ , then let  $S_{h_S+1}, \dots, S_h$  be simply “empty teeth”. We use  $|S_i|$  to represent the number of meta-modules stacked 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’ll 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  $i$ th tooth has length  $\max(|S_i|, |T_i|)$ ,  $1 \leq i \leq h$ . The common comb 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, Figures 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.



**Fig. 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 that of  $C_U$ .  $C_U$  consists of all shaded and unshaded boxes. Labels indicate which meta-modules were 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 missing meta-modules in the common comb. 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 the proper length, then  $C_S$ '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  $S_1, \dots, S_i$  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 is 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  $O(1)$  basic 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  $|S_i|$   $O.dequeue()$  { remove from  $O$  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  $O(n)$  atom 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  $O(1)$  atom operations, resulting in  $O(m) = O(n)$  parallel steps and atom operations.

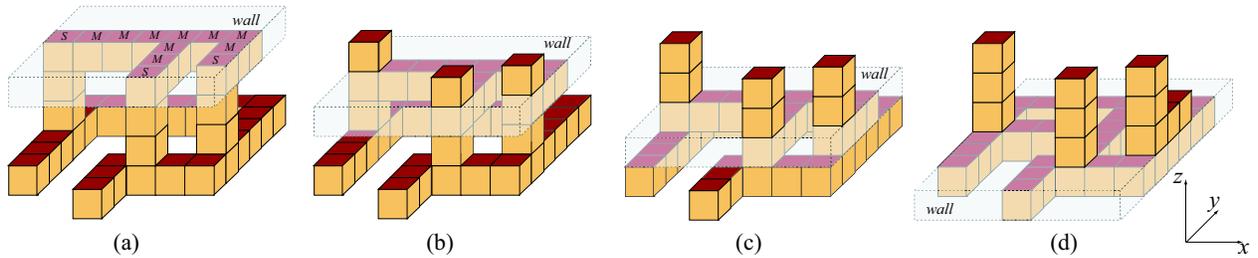
**Overall 2D Reconfiguration Algorithm** The general algorithm to reconfigure any  $m$  meta-module robot  $S$  to any other  $m$  meta-module robot  $T$  consists of four major steps: (1) reconfigure  $S$  into the comb  $C_S$ , (2) reconfigure  $C_S$  into the common comb  $C_{ST}$ , (3) reverse the moves of the 2D-COMB-TO-COMMON-COMB algorithm to reconfigure  $C_{ST}$  into target comb  $C_T$ , and (4) reverse the moves of the 2D-COMBING algorithm to reconfigure  $C_T$  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  $O(n)$  atom 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 target robot  $T$ . In transforming to the 3D common comb there are two intermediate configurations, a terrain configuration and a (regular) 3D comb configuration.

**Source Robot to 3D Terrain** We use the 3D analog of the 2D-COMBING process, 3D-COMBING, to reconfigure  $S$  into a 3D terrain as follows. The wall now consists of an entire 2D horizontal layer of meta-modules, initially the topmost single layer of  $S$ . 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. Unlike the 2D case, a 3D moving wall component here 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



**Fig. 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.

a layer, the moving components slide past the stationary meta-modules (using a  $\text{SLIDE}(z^-)$  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  $O(n)$  atom operations.*

*Proof.* Follows an argument similar to that in Lemma 2.

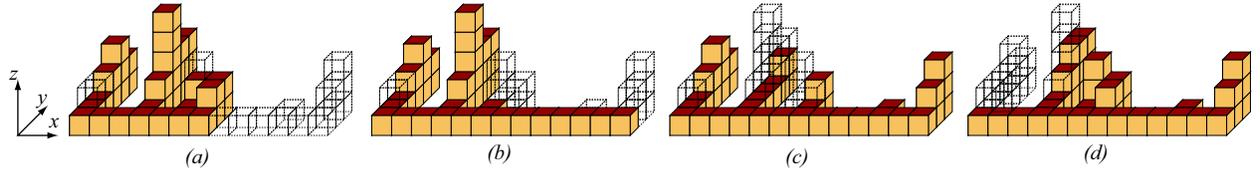
**3D Terrain to 3D Comb** A 3D Terrain  $I$  is reconfigured into a 3D comb by applying the 2D-COMBING algorithm of Section 3.1 to its base layer, thus reconfiguring the base layer into a 2D comb. As the base meta-modules move during the reconfiguration, they carry along the towers resting on top. If  $B(I)$  is the base of  $I$ , then a call to  $\text{2D-COMBING}(B(I))$  using the  $\text{SLIDE}$  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  $O(n)$  atom operations.*

**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 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  $j$ th 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. Since  $C_S^i$  and  $C_T^i$  may not contain the same number of meta-modules (as they did in Section 3.1), there may not even be enough meta-modules in  $C_S^i$  to fill the entire handle of  $C_U^i$ , in which case  $C_S^i$  will become only a portion of the handle that starts with module  $(i, 1, 1)$ . In Figure 6a,  $C_U$  consists of the solid and wireframe boxes; the solid boxes alone are  $C_S$  after Step 1 completes.



**Fig. 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  $C_S^i$  by tunneling meta-modules from  $C_S^4$ . (d) The 3D common comb (solid boxes only).

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, resulting in  $C_S$  and  $C_U$  having the same number of  $z$ -combs. Figure 6b shows the results of this step. Notice that the order of meta-modules in  $O$  ensures that each module moved is from the  $z$ -comb consisting of at least two meta-modules that is nearest the end of the  $xy$ -comb handle. For example, if  $C_S^x$  is the nearest  $z$ -comb, then all its teeth meta-modules and all but one (i.e.,  $(x, 1, 1)$ ) of its handle meta-modules (if needed) will be relocated to the  $xy$ -comb handle. If more meta-modules are needed, then they will be drawn from the next nearest  $z$ -comb of at least two meta-modules. The path the  $k$ -TUNNEL operations follow in this step is down a tooth in the nearest  $z$ -comb, through a (possibly length 0) handle section of a  $z$ -comb to the  $xy$ -comb handle, and through the  $xy$ -comb handle to its end. The order of the meta-modules in  $O$  ensures that each leg of this path is unattached to other meta-modules, thus allowing the tunnel move to be performed in  $O(1)$  time.

In Step 4, the teeth of each  $z$ -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 of Step 4,  $O$  contains exactly the teeth and handle meta-modules in combs  $C_S^1, \dots, C_S^i$  (with the exception of the handle meta-modules in  $C_S$ 's  $xy$ -comb, which were never in  $O$ ). 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 so 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 TUNNEL operation follows a path

whose segments are not attached to other meta-modules, allowing  $O(1)$  tunnel moves. A stage of Step 4 is illustrated in Figure 6c, with Figure 6d showing the resulting 3D common comb (solid meta-modules).

---

**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( $C_S^i, C_U^i$ ) (where the combs are parallel to the  $yz$  coordinate 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 from  $O$  meta-modules that are 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  $k \in \{3, 4\}$
- 

**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 major steps: (1) reconfigure  $S$  into a 3D terrain  $I_S$ , (2) reconfigure  $I_S$  into a 3D comb  $C_S$ , (3) reconfigure  $C_S$  into the common comb  $C_{ST}$ , (4) reconfigure  $C_{ST}$  into  $C_T$  by applying the moves of the 3D-COMB-TO-COMMON-COMB algorithm in reverse, (5) reconfigure  $C_T$  into the target 3D terrain  $I_T$  by applying the moves of the TERRAIN-TO-COMB algorithm in reverse, and (6) reconfigure  $I_T$  into the target  $T$  by applying the moves of the 3D-COMBING algorithm in reverse.

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

## 4 Parallel Implementation

The combing algorithm is parallel in that all moving components move simultaneously with the wall. But the 3D-COMB-TO-COMMON-COMB and 2D-COMB-TO-COMMON-COMB algorithms as presented are purely sequential. However, by modifying this algorithm slightly, some parallelization can be obtained.

For example, consider step 2 of Algorithm 2 which extends the handle of  $C_S$  to match the length of  $C_U$ 's handle. Let  $k$  be the number of meta-modules to be added to the end of  $C_S$ 's handle. Instead of tunneling one module at a time from the tooth nearest the end of  $C_S$ 's handle, we can tunnel in parallel  $k$  meta-modules, one from each of the last  $k$  non-empty teeth. Specifically, a module from each of the last  $k$  non-empty teeth is compressed into the handle module at the base of each tooth. The meta-modules are then simultaneously uncompressed, pushing all  $k$  meta-modules to the end of the handle. Of course if there are only  $k' < k$  teeth

available to draw meta-modules from, then at most  $k'$  teeth can be moved in parallel. Since the number of atom operations performed in tunneling the  $k$  meta-modules is proportional to  $k$ , the overall algorithm stays linear in the number of atom operations.

Step 3 of Algorithm 2 can be parallelized similarly. Suppose for example the rightmost  $k$  teeth are too short. We can in parallel tunnel meta-modules to each of these teeth, drawing meta-modules from  $k$  non-empty teeth immediately to their left. Let  $I_t$  be the set of  $k$  rightmost short teeth and  $O_t$  be the first set of  $k$  nonempty teeth to their left. In parallel, the top module of each tooth in  $O_t$  is moved down to the handle, then along the handle to the base of a tooth in  $I_t$ , and then up to the top of the tooth. In each parallel step,  $k$  should be selected so that it is as large as possible under the constraint that there are  $k$  non-empty teeth to the left of the  $k$  rightmost short teeth.

Similar parallelizations can be achieved in the 3D as well by moving meta-modules in parallel through the handle of the  $xy$ -comb.

There are of course worst case configurations in which no parallelism is possible using these methods. However, the mechanical operations of the atom prototypes are slow enough that in practice it is worth trying to achieve even limited parallelism.

## 5 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 strikes 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 using information on its current state (moving or stationary), or by 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.

The COMB-TO-COMMON-COMB algorithms can also be distributed, albeit with some stronger requirements. First, the initial and final configurations  $S$  and  $T$  must be communicated to each 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 any algorithm of complexity  $O(n)$  in  $O(n)$  time. These requirements are necessary because each meta-module must initially run 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.

For the reverse of the COMBING algorithm taking  $C_T$  to  $T$ , if a meta-module were to run 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 do only  $O(n)$  processing. We describe it in 2D; the 3D algorithm is analogous. It assumes the meta-modules have the same processing power required by the COMB-TO-COMMON-COMB algorithms and that each meta-module knows its location in  $C_T$  and knows the final configuration  $T$ .

Using  $T$  and its  $(x, y)$  location, 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  $C_T$ . 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 have no way of knowing that they moved up a row. However, each meta-module can initially calculate 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  $O(n)$  atom operations.

The total processing time needed to determine the atom operations reduces to  $O(n)$  parallel time in the distributed implemented. 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. In the forward and reverse COMB-TO-COMMON-COMB algorithms, in parallel each atom runs the algorithm to precompute their moves, taking  $O(n)$  parallel time. The reverse COMBING algorithms require  $O(n)$  parallel time, as outlined above.

**Acknowledgments.** 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

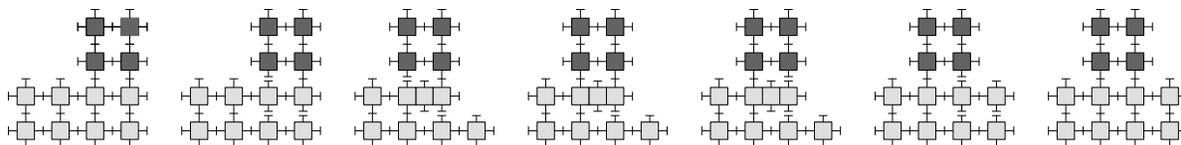
- [ABMP06] E.M. Arkin, M.A. Bender, J.S.B. Mitchell, and V. Polishchuk. The snowblower problem. In *Proc. 7th Intl. Workshop on the Algorithmic Foundations of Robotics*, 2006.
- [BFR02] Z. Butler, R. Fitch, and D. Rus. Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting. *IEEE/ASME Trans. on Mechatronics*, 7(4):418–430, 2002.
- [BR03] Z. Butler and D. Rus. Distributed planning and control for modular robots with unit-compressible modules. *The Intl. Journal of Robotics Research*, 22(9):699–715, 2003.
- [Hac07] T. Hackl. Personal communication, 2007.
- [RV01] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.
- [SHY02] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 4095–4101, 2002.
- [VYS02] S. Vassilvitskii, M. Y., and 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*, pages 117–122, 2002.

## Appendix

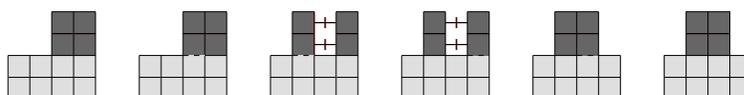
This appendix shows how to achieve the SLIDE and  $k$ -TUNNEL moves in both the expanded and the contracted models, using meta-modules of minimum size. In the expanded (contracted) model, the atoms have their faces expanded (contracted) except when they are involved in a meta-module operation. Both models have been considered in the robotics community, and one may be preferred over another 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. Here we show that  $2 \times 2 \times 2$  meta-modules suffice: this result is due to T. Hackl [Hac07]. As for the contracted model, the sequences of atom operations that we show have been used in previous reconfiguration algorithms [VYS02], although we offer a modified version for the  $k$ -TUNNEL move that avoids 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.

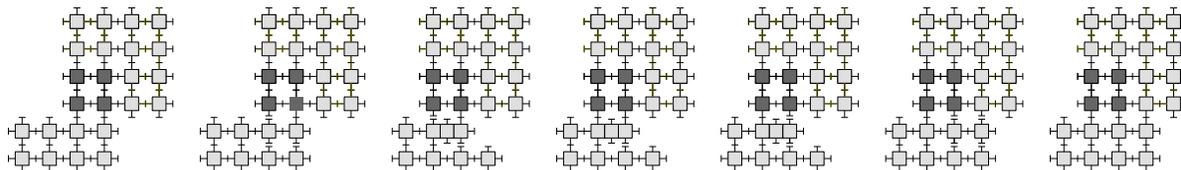


**Fig. 7.** Expanded robot. First steps of SLIDE applied to the top meta-module. Only one layer shown.

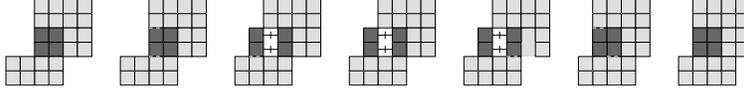


**Fig. 8.** Contracted robot. First steps of SLIDE applied to the top meta-module. Only one layer shown.

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



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



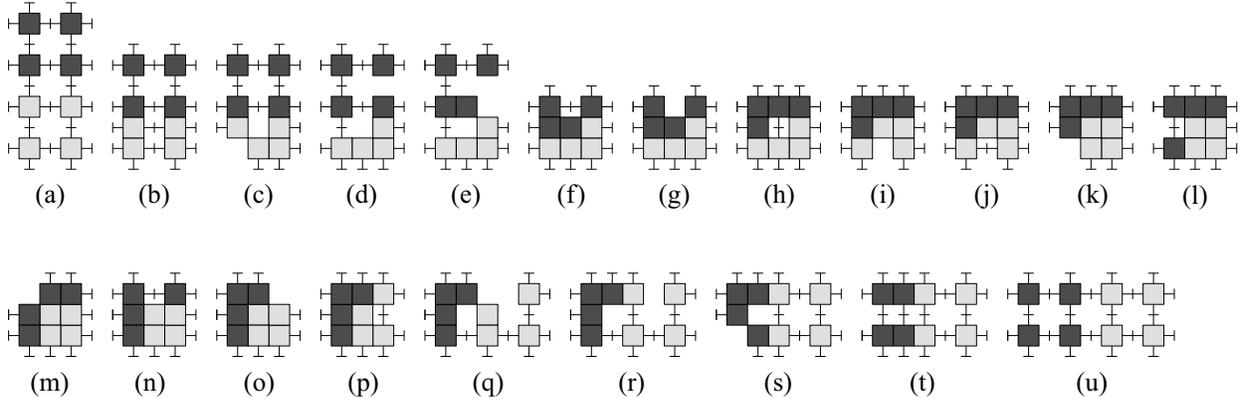
**Fig. 10.** Contracted robot. Carrying meta-modules in a SLIDE move. Only one layer shown.

*Connectedness of the robot.* Notice that sliding a 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 atom 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

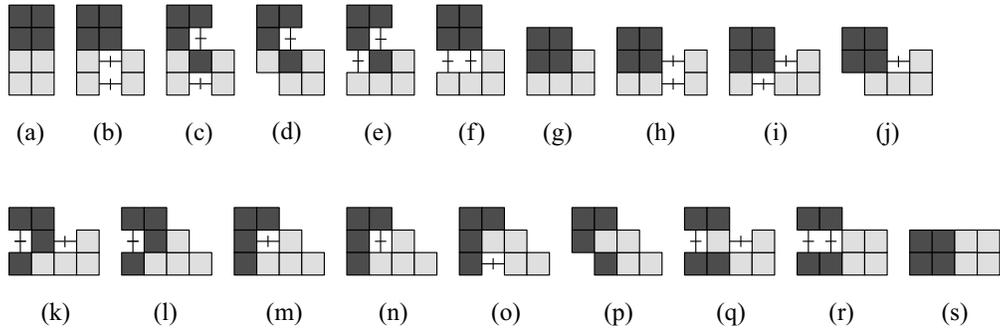
Figures 11 and 12 show the 1-TUNNEL( $(x, y + 1), (x + 1, y)$ ) atom operations in the expanded and contracted models. Figures 13 and 14 show selected steps of 1-TUNNEL( $(x, y + 3), (x + 3, y)$ ) in both models.



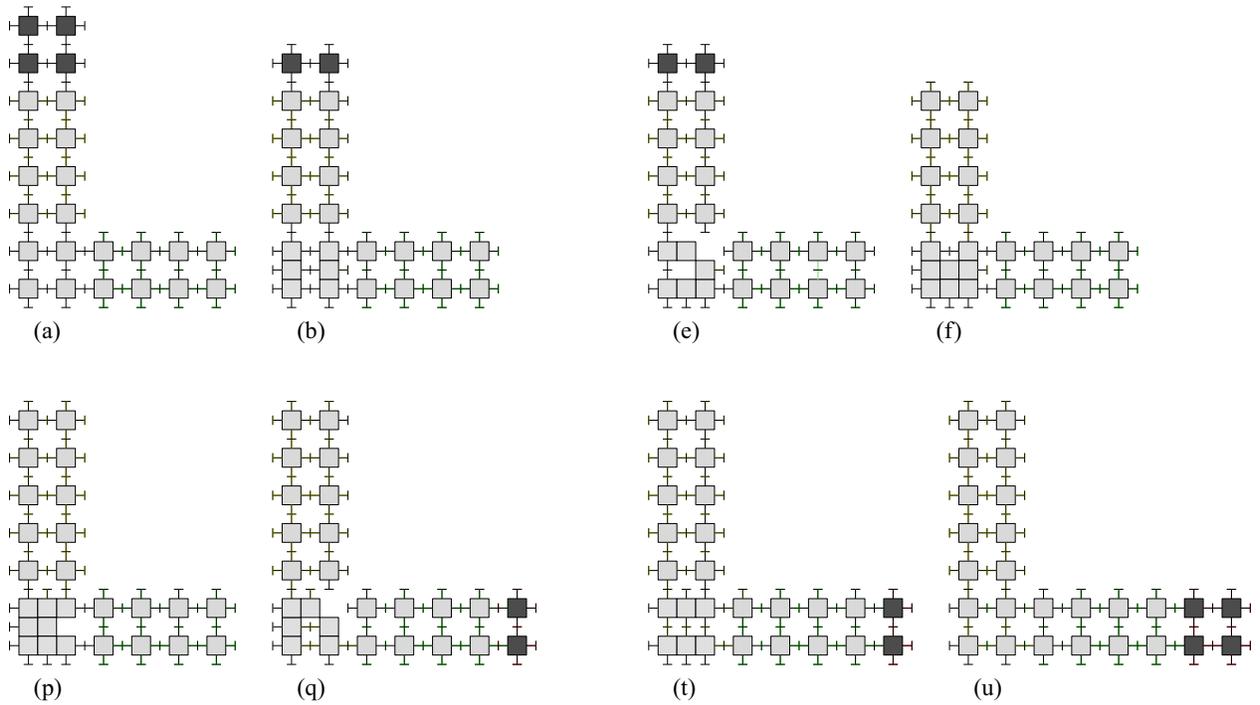
**Fig. 11.** Expanded robot. 1-TUNNEL( $(x, y + 1), (x + 1, y)$ ). Only one layer shown. Attachments and detachments are not shown.

*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.

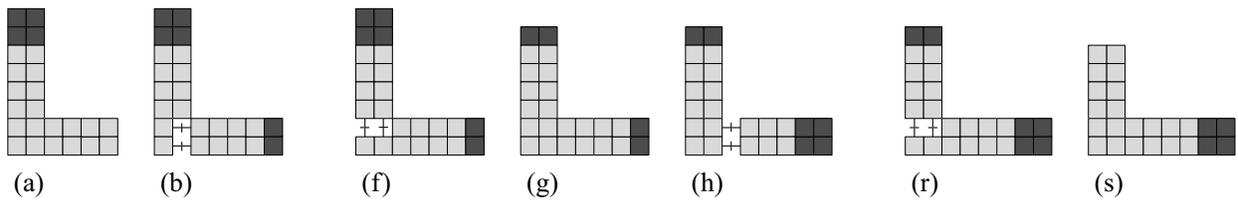
*Complexity of the move.*  $k$ -TUNNEL is implemented in  $O(k)$  parallel steps using  $O(k)$  atom operations, as long as there are no meta-modules attached along the paths between consecutive turns, as is the case in our algorithms here. When there are attached meta-modules,  $k$ -TUNNEL can still be implemented in  $O(k)$  parallel steps, but the number of atom 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.



**Fig. 12.** Contracted robot. Example of  $1\text{-TUNNEL}((x, y + 1), (x + 1, y))$ . Only one layer shown. Attachments and detachments are not shown.



**Fig. 13.** Expanded robot. Selected steps of  $1\text{-TUNNEL}((x, y + 3), (x + 3, y))$ . Labels refer to Figure 11.



**Fig. 14.** Contracted robot. Selected steps of  $1\text{-TUNNEL}((x, y + 3), (x + 3, y))$ . Labels refer to Figure 12.